Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 2957

A Multi-Level Application Isolation Framework for Peer-to-Peer Desktop Grid Platforms

Robert Hanussek

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Kurt Rothermel
Betreuer:	Dipl.-Inform. Sven Schulz, PD Dr. Wolfgang Blochinger
begonnen am:	2. September 2009
beendet am:	4. März 2010
CR-Klassifikation:	C.2.4, C.4, D.2.11, D.4.6, G.1.6

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Desktop-Grid-Plattformen haben sich von monolithischer Software zu voll-konfigurierbaren Micro-Kernel-Systemen entwickelt, die Peer-to-Peer-Netzwerke bilden [SBHD08]. Diese Systeme erlauben die parallele Ausführung unterschiedlicher Desktop-Grid-Anwendungen auf einem Computer und vereinheitlichen gleichzeitig deren Verwaltung. Diese Zusammenführung stellt jedoch auch ein Risiko für die Robustheit und Sicherheit der Anwendungen dar. Die Gefahr, dass sich Anwendungen gegenseitig beeinflussen, wird durch eine gemeinsame Plattform erhöht. Dadurch entstehen neue Anforderungen an die Isolation der Anwendungen. Neben dem Schutz des Wirtsystems vor Schadprogrammen rückt der Schutz der Anwendungen untereinander in den Fokus. Organisationseigene Anwendungen sollen z.B. von Manipulationen fremder Anwendungen abgeschirmt werden. Ein andere Anwendungsfall ist die Isolation inkompatibler Anwendungen, die den Betrieb gegenseitig stören.

Die, in diesem Dokument, vorgestellte Software bietet Funktionalität zur Anwendungsisolation, um derartige Szenarien zu unterstützen. Die Software wurde auf Basis von OSGi [OSG09a], einem Standard für modulare Software, entwickelt. Er setzt bereits eine Isolation der Module, nach dem Prinzip des Information Hiding [PCW89], voraus. Darüber hinaus ermöglicht die Software die Isolation von Anwendungen mit Hilfe unterschiedlicher Methoden. Denkbare Isolationsstufen beginnen bei der, durch das OSGI-Framework gegebenen, Isolation und reichen über JVM-unterstützte Isolation und Prozesse bis hin zur Systemvirtualisierung. Der Kommunikationsaufwand, sowie die zur Verfügung stehende Rechenleistung, hängen stark vom Isolationsgrad der Anwendungen ab. Deshalb kann der Isolationsgrad unter Berücksichtigung des jeweiligen Anwendungsfalls gewählt werden, so dass eine Abwägung zwischen Sicherheit und Leistung möglich ist.

Im folgenden Abschnitt 1.2 werden Beweggründe für die Isolation von Anwendungen vorgestellt. In 1.3 sind einige konkrete Anwendungsfälle beschrieben.

1.2 Beweggründe für die Isolation von Anwendungen

Für die Isolation einer Anwendung kann es unterschiedliche Gründe geben. Einerseits ist die Sicherheit des Wirtsystems, auf dem die Anwendung ausgeführt wird, zu wahren. Ande-

rerseits kann das Wirtsystem selbst bereits Schadprogramme enthalten, die die Sicherheit der Anwendung beeinträchtigen. Auch kann es sich bei einer anderen Anwendung um ein Schadprogramm handeln, das das Wirtsystem und die Desktop-Grid-Anwendungen gleichermaßen gefährdet.

Unter *Sicherheit* wird in dieser Arbeit die Fähigkeit eines Computersystems verstanden, die Vertraulichkeit und Integrität der Daten sowie die Verfügbarkeit der Dienste zu gewährleisten [ALRL04]. Diese Eigenschaften sind auch als *CIA Triade* bekannt, abgeleitet von den Anfangsbuchstaben der englischen Begriffe Confidentiality, Integrity und Availability.

Die Möglichkeit der Interaktion zwischen einer Desktop-Grid-Anwendung und dem Wirtsystem oder einer anderen Anwendung stellt ein Risiko für die Sicherheit der Beteiligten dar. Als *Isolation* werden Maßnahmen bezeichnet, die solchen Interaktionen entgegenwirken.

Unter dem Begriff *Wirtsystem* werden alle parallel zur Desktop-Grid-Anwendung laufenden Programme verstanden, die nicht auf der Desktop-Grid-Plattform aufsetzen, also selbst keine Desktop-Grid-Anwendungen sind. Darunter fallen also alle Prozesse des Betriebssystems, die Java-Laufzeitumgebung sowie die Desktop-Grid-Plattform selbst.

Zur Wahrung der Sicherheit der Anwendungen und des Wirtsystems kommen also folgende Beweggründe in Frage:

- Isolation zum Schutz des Wirtsystems vor den Anwendungen
- Isolation zum Schutz der Anwendungen vor dem Wirtsystem
- Isolation zum Schutz der Anwendungen voreinander

Neben Sicherheitsaspekten kann es noch andere Motive für die Isolation einer Anwendung geben. Es können damit Schwächen in der Architektur des Wirtsystems umgangen werden. Sind bestimmte Ressourcen des Systems nur für die exklusive Nutzung einer Anwendung konzipiert worden, kann es bei der Ausführung mehrerer Anwendungen, die diese Ressource benötigen, zu Konflikten kommen. Bei den statischen Attributen `System.in`, `System.out` und `System.err` einer Java Laufzeitumgebung handelt es sich beispielsweise um solche Ressourcen. Sie erlauben den Zugriff auf die Standardeingabe und -ausgabe [Sun06b]. Leitet eine Anwendung die Standardausgabe um, betrifft dies auch die anderen Anwendungen, die in der Laufzeitumgebung ausgeführt werden. Durch Isolation der Anwendungen, z. B. mittels Ausführung in einer eigenen Laufzeitumgebung, kann diese Schwäche umgangen werden.

Auch den Schwächen einer Anwendung kann durch Isolation Abhilfe geschaffen werden. Setzen die Anwendung z. B. unterschiedliche Betriebssysteme voraus, können sie, voneinander isoliert, in virtuellen Maschinen ausgeführt werden, die das jeweilige Betriebssystem bereitstellen. Es kann also die Portabilität von Anwendungen verbessert werden. Das ist insbesondere bei Desktop-Grid-Systemen ein Vorteil, da ihnen oft ein heterogene Infrastruktur zu Grunde liegt.

Betriebssystem bieten meist Werkzeuge zur Überwachung der Auslastung der CPU oder des Arbeitsspeichers. Diese Werkzeuge, z. B. `top` [Linb] oder `ps` [Lina], arbeiten jedoch auf Prozessebene. Werden in einem Prozess mehrere Anwendungen ausgeführt, wie es bei OSGi üblich ist, lassen sich deren Ressourcen nicht einzeln betrachten. Zwar bieten einige Werkzeuge auch die Überwachung von Threads an, jedoch lassen sich diese oft schwer einer bestimmten Anwendung zuordnen. Zudem muss nicht für jede Anwendung ein eigener Thread existieren. Auch die Kontrolle der Ressourcen mit Werkzeugen der Betriebssysteme, z. B. die Beschränkung des nutzbaren Arbeitsspeichers mit `ulimit` [Linc], findet meist auf Prozessebene statt. Eine Isolation der Anwendungen, so dass sie in eigenen Prozessen ausgeführt werden, ermöglicht eine Überwachung und Kontrolle der Ressourcen, die von einer Anwendung konsumiert werden.

Neben Sicherheitsaspekten gibt es also noch weitere Beweggründe für die Isolation von Anwendungen:

- Isolation zur Umgehung von Schwächen in der Architektur des Wirtsystems
- Isolation zur Umgehung von Schwächen in der Architektur der Anwendung
- Isolation zur Unterstützung einer feineren Ressourcenüberwachung und -kontrolle

1.3 Anwendungsfälle

Im Folgenden werden einige konkrete Anwendungsfälle für die Isolation von Anwendungen beschrieben.

1.3.1 Öffnung eines Desktop-Grid-Netzwerks

Eine Universität möchte ihr Desktop-Grid-Netzwerk, bestehend aus den Computer-Pools für Studenten und Arbeitsplätzen der Mitarbeitern, auch anderen Forschungsorganisationen und Unternehmen gegen Entgelt zur Verfügung stellen. Bisher wurde das Desktop-Grid nur von Mitarbeitern der Universität genutzt. Die Ausführung fremder Anwendungen stellt ein Risiko dar, da deren Quelltext nicht eingesehen werden kann oder dieser zu umfangreich für eine Überprüfung ist. Unter Umständen herrschen in den anderen Organisationen auch niedrigere Qualitätsstandards, wodurch sich leichter Fehler in die Software einschleichen. Solche Fehler können die Verfügbarkeit der Rechnerinfrastruktur gefährden, z. B. durch übermäßigen Ressourcenverbrauch.

Die Universität beschließt alle fremden Anwendungen zunächst isoliert in einer eigenen Virtuellen Maschine auszuführen, um das Sicherheitsrisiko zu verringern. Über einen bestimmten Zeitraum werden die Anwendungen überwacht. Werden keine Auffälligkeiten, wie

z. B. ein vorzeitiges Beenden der Laufzeitumgebung (Absturz) oder ein zu hoher Ressourcenverbrauch, beobachtet, können die Anwendungen in einer niedrigeren Isolationsstufe ausgeführt werden.

1.3.2 Nutzung eines öffentlichen Desktop-Grids

Ein Unternehmen des öffentliche Personennahverkehrs möchte die Einsatzplanung der Fahrer in ein externes Desktop-Grid auslagern. Da es sich dabei um datenschutzrechtlich sensible Daten handelt, ist die Vertraulichkeit der Anwendung als besonders hoch einzuschätzen. Um die Daten zu schützen läuft die Kommunikation zwischen Desktop-Grid-Clients nur über sichere Kanäle. Jedoch werden auf den Wirtsystemen auch Desktop-Grid-Anwendungen fremder Organisationen ausgeführt. Um die Anwendung vor unberechtigten Zugriffen zu schützen, stellt das Unternehmen die Isolationsvorgabe, dass ihre Anwendung in einer *Closed-Box Virtual Machine* auszuführen ist. Eine solche Closed-Box VM nutzt Trusted Computing Technologie, um einen isolierten Bereich innerhalb des Wirtsystems zu erstellen, der nicht einmal mehr dem Administrator des Wirtsystems zugänglich ist [MMJZ09].

1.3.3 Parallele Ausführung inkompatibler Anwendungen

Zwei Anwendungen, die die Standardausgabe (`System.out`) zur Protokollierung von Warn- und Fehlermeldungen nutzen, sollen auf dem selben Wirtsystem ausgeführt werden. Beide Anwendungen leiten die Standardausgabe in eine Log-Datei um, so dass die Meldungen auch nach Beendigung der Ausführung noch verfügbar sind. Eine Ausführung der Anwendungen in der selben Java-Laufzeitumgebung bewirkt jedoch, dass die Meldungen beider Anwendung in die selbe Datei umgeleitet werden. Die Meldungen werden in die Datei der Anwendung ausgeben, die zuletzt die Methode `System.setOut` zur Einrichtung der Umleitung aufgerufen hat. Die gespeicherten Meldungen sind dann nur noch schwer einer Anwendung zuzuordnen. Durch die isolierte Ausführung der Anwendungen in eigenen Prozessen, und damit eigenen Java-Laufzeitumgebungen, steht jeder Anwendung eine eigene Standardausgabe zur Verfügung.

1.3.4 Verbesserung der Portabilität einer Anwendung

Zwei Anwendungen, die nur zu bestimmten Java-Laufzeitumgebungen kompatibel sind, sollen auf dem selben Wirtsystem ausgeführt werden. Dies kann durch die isolierte Ausführung der Anwendungen in eigene Prozesse bewerkstelligt werden, die die jeweils passenden Laufzeitumgebungen nutzen.

Auch bei Anwendungen, die für unterschiedliche Betriebssysteme entwickelt wurden, schafft eine Isolation der Anwendungen Abhilfe. Eine getrennte Ausführung in virtuellen Maschinen erlaubt die Ausführung beider Anwendungen auf dem gleichen Wirtsystem.

Gliederung

In Kapitel 2 werden die Grundlagen dieser Arbeit beschrieben. Kapitel 3 stellt verwandte Arbeiten vor. Der Aufbau des Systems sowie die Annahmen, die bei dessen Entwicklung getroffen wurden, werden in Kapitel 4 beschrieben. In Kapitel 5 werden unterschiedliche Isolationsarten vorgestellt und die Effektivität und Effizienz von zwei Arten untersucht. Aus den Vorgaben zur Isolation, die z. B. vom Benutzer des Wirtsystems gestellt werden, muss eine angemessene Isolationsart für jede Anwendung abgeleitet werden. Mit diesem Problem beschäftigt sich Kapitel 6. Kapitel 7 fasst die Arbeit zusammen und stellt Anknüpfungspunkte vor.

2 Grundlagen

In diesem Kapitel werden grundlegende Themen vorgestellt, die für das Verständnis des weiteren Inhalts von Bedeutung sind.

2.1 Desktop Grid Computing

Als *Grid Computing* wird die Durchführung von verteilte Berechnungen auf einem Zusammenschluss aus unabhängig arbeitenden Computern bezeichnet. Das durch den Zusammenschluss entstandene *Grid* ermöglicht die transparente Nutzung heterogener Ressourcen, die unterschiedlichen Organisation angehören [CKB+07]. Durch die Vereinigung vieler einzelner Computer steht eine hohe Rechenleistungen zur Verfügung. Der Einsatz von Hardware, die im Vergleich zu Supercomputern oder spezialisierten Clustern mit vergleichbarer Leistung kostengünstig ist, erlaubt es, den finanzielle Aufwand niedrig zu halten.

Ein *Desktop Grid* besteht, im Gegensatz zu konventionellen Grids, aus Computern, die nicht ausschließlich dem Grid zur Verfügung stehen. Die Ressourcen werden vornehmlich anderweitig genutzt, z. B. als Büro- oder Privat-PC. Die Leerlaufzeiten dieser Computer soll für rechenintensive Anwendungen nutzbar gemacht werden, indem diese brachliegenden Ressourcen in einem Grid zusammengefasst werden.

Ein bekanntes Beispiel ist SETI@home, eine Anwendung, die aufgezeichnete Funksignale nach Hinweisen auf intelligentes außerirdisches Leben untersucht. Als Middleware, im Folgenden *Plattform* genannt, wird die Berkeley Open Infrastructure for Network Computing (BOINC) eingesetzt [And04]. Sie stellt die Dienste zur verteilen Ausführen von Anwendungen im Grid bereit.

Bei BOINC/SETI@home handelt es sich auch um ein Beispiel für *Volunteer Computing*. Freiwillige stellen ihre Ressourcen dem Desktop Grid zur Verfügung ohne eine finanzielle Entlohnung dafür zu erhalten. In *Enterprise Desktop Grids* hingegen werden die Computer an den Arbeitsplätzen einer Organisation zur verteilten Berechnung genutzt. Beide Ansätze haben gemeinsam, dass die Ressourcen vornehmlich einem anderen Zweck dienen und dadurch keine gleichbleibende Rechenleistung liefern. Es ist nicht zuverlässig vorherzusehen, wann ein Computer zur Verfügung steht. Diese Unbeständigkeit der Rechenleistung stellt eine große Schwierigkeit bei der Nutzung der Ressourcen eines Desktop Grids dar.

Das Wirtsystem soll dem Benutzer weiterhin als Desktop-Computer zur Verfügung stehen, d. h. es muss eine flüssige Interaktion mit den Wirtsystem möglich sein. Die Auslastung durch das Grid Computing muss auf die vom Nutzer geforderte Leistung abgestimmt werden.

Eine weitere Herausforderung bildet die heterogene Umgebung eines Desktop Grids. Da unterschiedlichste Computer in einem Grid zusammengefasst sind, werden hohe Anforderungen an die Portabilität der Plattform und der Anwendungen gestellt. Zudem sollte der Aufwand für die Installation der Plattform und der Anwendungen möglichst gering gehalten werden. Da sonst die Bereitschaft zur Beteiligung am Grid sinkt. Das gilt insbesondere für das Volunteer Computing.

Da keine zentrale Kontrolle über die Ressourcen eines Desktop Grids existiert, können durch schädliche Anwendungen Sicherheitsrisiken für die Inhaber der Ressourcen entstehen. Aber auch die Nutzer der Desktop Grids sehen sich Gefahren ausgesetzt, wie z. B. der Manipulation der Ergebnisse einer Berechnung durch den Anbieter einer Ressource. Weitere Sicherheitsrisiken sind in Abschnitt 1.2 aufgeführt.

2.1.1 Cohesion

Bei Cohesion handelt es sich um eine Plattform für Peer-to-Peer Desktop Grids [SBHD08]. Das in dieser Arbeit vorgestellte Isolationssystem ist als Teil dieser Plattform konzipiert.

Zielsetzung von Cohesion ist eine umfangreiche Unterstützung für die verteilte Berechnung von *Irregular Structured Problems* (ISPs). Im Gegensatz zu *Embarrassingly Parallel Problems* lassen sich ISPs nur mit hohem Aufwand in parallele Aufgaben aufteilen. Die Aufteilung geschieht dynamisch, während der Berechnung, wodurch zusätzlicher Kommunikationsaufwand entsteht. Cohesion verwendet dazu eine verteile Aufgabenverwaltung. Für die Kommunikation werden Peer-to-Peer-Protokolle eingesetzt, um zentrale Engpässe zu vermeiden.

Für die Berechnung von ISPs existieren viele Ansätze, die sich bis auf grundlegende Dienste, wie die Kommunikation, voneinander unterscheiden können. Es ist als nicht möglich eine Plattform zu entwickeln, deren Funktionalität alle Ansätze abdeckt. Stattdessen stellt Cohesion Standard-Funktionalität für wiederkehrende Aufgaben zur Verfügung, die durch anwendungs-spezifische Komponenten erweitert oder komplett ersetzt werden kann. Cohesion implementiert dazu das Microkernel-Entwurfsmuster. Zur Umsetzung des Entwurfsmuster kommt OSGi zum Einsatz, das im nächsten Abschnitt vorgestellt wird. OSGi baut auf Java-Technologie auf, die eine betriebssystemunabhängige Implementierung ermöglicht. Dadurch wird der Aktionsradius der Plattform auf alle gängigen Betriebssystem ausgeweitet, die eine Java-Laufzeitumgebung bereitstellen können. Die hohe Portabilität stellt einen großen Vorteil in heterogenene Desktop Grids dar.

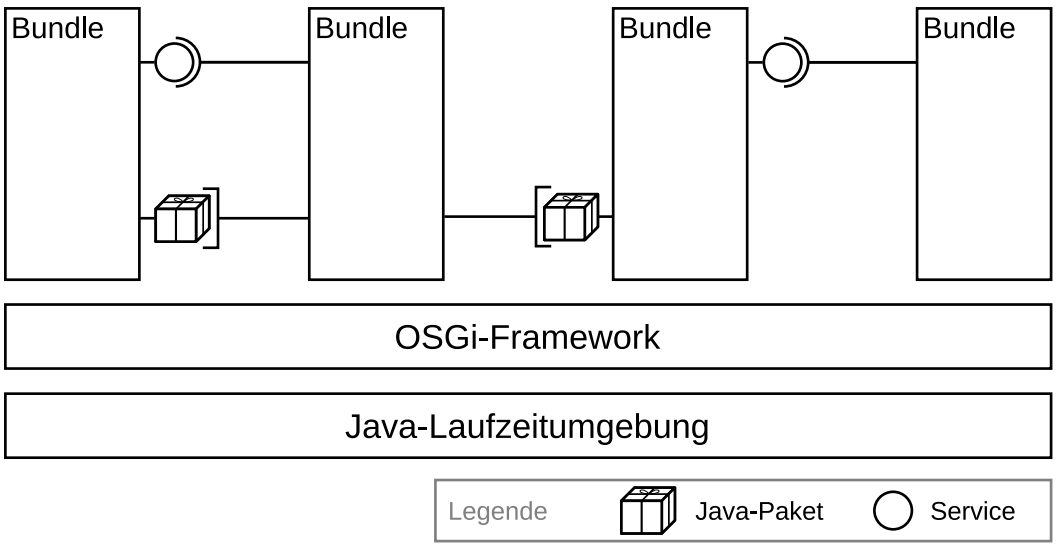

Abbildung 2.1: OSGi-Framework

2.2 OSGi

Im März 1999 wurde die *Open Services Gateway Initiative* (OSGi) gegründet, mit dem Ziel ein System zu entwerfen, das die Entwicklung und Verteilung von Software vereinfachen soll. Der Fokus lag damals besonders auf mobilen Geräten mit begrenztem Arbeitsspeicher [Opeoo]. Inzwischen wurde die Abkürzung OSGi zum Eigennamen erhoben. Die *OSGi Alliance* umfasst prominente Mitglieder, wie z. B. IBM, Oracle, Siemens und die Deutsche Telekom. Der Schwerpunkt liegt heute nicht mehr nur auf eingebetteten Systemen. Auch auf dem Computer zu Hause oder im Büro soll OSGi eingesetzt werden. Der Grundgedanke ist dabei derselbe geblieben: Durch Modularisierung, also der genauen Definition von Modulen und deren Schnittstellen, soll die Verteilung und Verwaltung von Programmen erleichtert werden. Auch soll durch die Modularisierung die Wartbarkeit und Austauschbarkeit von Programmkomponenten gewährleistet werden.

Von der OSGi Alliance werden Spezifikationen für ein solches Modulsystem veröffentlicht, das als *OSGi Framework* bezeichnet wird. Die Aspekte des Framework werden in der Spezifikation in unterschiedliche Schichten (engl. Layer) aufgeteilt [OSGo9a]. Im Folgenden werden die vier Schichten und deren Funktionalität beschrieben.

2.2.1 Module Layer

Im Module Layer werden die Module des OSGi Frameworks definiert, die *Bundles*. Ein Bundle besteht aus Java Bytecode und Metadaten zur Beschreibung des Bundles. Es können auch

zusätzlichen Ressourcen, wie z. B. Konfigurationsdateien oder native Programmbibliotheken enthalten sein. Implementiert wird ein Bundle als komprimierbare Archivdatei im Java-Archive-Format (JAR), das auch für Standard-Java-Programme zum Einsatz kommt. Die Metadaten sind in der Manifest-Datei, die in jedem JAR-Archiv enthalten ist, gespeichert.

In Java wird ausführbarer Code in *Klassen* zusammengefasst. Klassen können wiederum in *Paketen* (engl. packages) zusammengefasst werden. In einem Bundle sind Java-Pakete enthalten. Diese Pakete sind für andere Bundles nicht zugänglich. Ein Bundle kann jedoch Java-Pakete *exportieren*. Die Pakete stehen dann auch anderen Bundles zur Verfügung. Ein anderes Bundle kann die Java-Pakete importieren und den enthaltenen Code nutzen. Das Anfordern eines Pakets für den Import geschieht mit Hilfe des Paketnamens, nicht mit dem Namen des exportierenden Bundles. Dadurch ist das exportierende Bundle, das das Paket zur Verfügung stellt, leicht austauschbar.

Wie die Isolation der Java-Pakete der Bundles umgesetzt wird, ist in Abschnitt 5.1 beschrieben.

Import sowie Export eines Java-Pakets müssen, durch Angaben in der Manifest-Datei des Bundles, explizit veranlasst werden. Im Ursprungszustand bestehen zwischen Bundles also keine Abhängigkeiten. Sie müssen durch den Entwickler erst hergestellt werden. Dadurch wird das Prinzip der losen Kopplung gefördert. Abhängigkeiten müssen einerseits bewusst hergestellt werden und sind andererseits aus den Metadaten des Bundles ersichtlich. Das erleichtert die Wartung und den Austausch der Bundles.

Beim Export und Import können die Paketnamen mit Versionsnummern versehen werden. Damit steht ein Mittel zur Verfügung Inkompatibilitäten durch Änderungen von Paketen zu vermeiden. Ein importierenden Bundle verlangt eine bestimmte Version eines Pakets. Wird Code in einem exportierten Paket geändert, wird dies durch eine Änderungen der Versionsnummer deutlich gemacht. Dem importierenden Bundle wird die geänderte Version nur zur Verfügung gestellt, wenn sie verlangt wurde. Zudem können mehrere Versionen eines Pakets gleichzeitig zur Verfügung gestellt werden und damit Bundles auf unterschiedlichen Entwicklungsständen unterstützt werden.

Aufgabe eines OSGi-Frameworks ist es, die Abhängigkeiten zwischen Bundles aufzulösen, d. h. ein exportiertes Java-Paket eines Bundles anderen Bundles, die das Paket benötigen, zugänglich zu machen. In der Terminologie von OSGi wird dieser Vorgang *Verdrahtung* (engl. wiring) genannt. Das Framework verdrahtet importierende Bundles mit exportierenden Bundles unter Berücksichtigung von Bedingungen, wie der Versionskompatibilität.

2.2.2 Service Layer

Services stellen die Kommunikationsschnittstellen zwischen Bundles, den Modulen von OSGi, dar. Ein Bundle veröffentlicht einen Service unter Angabe einer Schnittstellendefinition.

Angefordert werden Services ebenfalls unter Angabe einer Schnittstellendefinition. Das OSGi-Framework liefert die Services, die die angeforderte Schnittstelle bieten. Dadurch bleibt der Serivce-Konsument von bestimmten Implementierungen unabhängig. Es kann jede Service-Implementierung genutzt werden, die der angeforderten Schnittstellendefinition entspricht.

Eine Schnittstellen wird in OSGi durch ein Java-Interface definiert. Ein Service ist ein gewöhnliches Java-Objekt, das das Interface implementiert. Ein Bundle registriert dieses Java-Objekt unter Angabe der implementierten Interfaces. Andere Bundles können einen Service unter Angabe des gewünschten Interfaces von der Service Registry anfordern. Genutzt wird der Service wie jedes andere Java-Objekt. Die Funktionen eines Services können also durch einfache Methodenaufrufe genutzt werden.

Services müssen explizit veröffentlicht werden. Ohne die Registrierung sind sie für andere Bundles also nicht sichtbar. Dadurch wird das Prinzip des Information Hiding [PCW89] umgesetzt.

Im Gegensatz zu Java-Paketen können Services zu einem beliebigen Zeitpunkt, während der Laufzeit eines Bundles, registriert werden und die Registrierung auch wieder rückgängig gemacht werden.

Ein Service besitzt Metadaten, sogenannte *Service Properties*. Ein Property besitzt einen Namen und einen Wert.

2.2.3 Life Cycle Layer

Im Life Cycle Layer ist definiert, wie Bundles, die Module von OSGi, installiert, aktualisiert und deinstalliert werden. Ist ein Bundle installiert, werden seine exportierten Java-Pakete bereits anderen Bundles zur Verfügung gestellt. Weitere Operationen zur Steuerung des Lebenszyklus sind das Starten und Stoppen von Bundles. Nachdem ein installiertes Bundle gestartet wurde, wird auch sein Code ausgeführt und es kann Services veröffentlichen und selbst Services anderer Bundles nutzen.

Die OSGi Alliance beschreibt ihr System als „Das dynamische Modulsystem für Java“ (engl. „The Dynamic Module System for Java“) [OSG]. Das Modulsystem wird als dynamisch bezeichnet, weil die oben beschriebenen Operationen zur Laufzeit ausgeführt werden können. Einem OSGi-basierten System können also zur Laufzeit Module, in Form von Bundles, hinzugefügt werden. Zudem können Komponenten entfernt werden, wenn sie nicht mehr benötigt werden. Das hat den Vorteil, das auch bei der Erweiterungen oder Aktualisierungen ein unterbrechungsfreier Betrieb der Systems gewährleistet werden kann.

Die Operationen zur Steuerung des Lebenszyklus können von jedem Bundle genutzt werden. Jedoch definiert die OSGi-Spezifikation Berechtigungen mit deren Hilfe festgelegt werden kann, welche Bundles z. B. Installationen vornehmen dürfen.

2.2.4 Security Layer

Im Security Layer werden die Sicherheitsmechanismen von OSGi definiert, die auf dem Berechtigungsmodell von Java aufbauen. Es werden zusätzliche Berechtigungen definiert, die die Kontrolle über Interaktionen zwischen Bundles, den Modulen von OSGi, ermöglicht. So kann z. B. festgelegt werden, welche Services ein Bundle nutzen oder veröffentlichen darf.

Um an Bundles Berechtigungen zu vergeben und um sie ihnen entziehen zu können, müssen sie authentifiziert werden. In OSGi stehen dazu zwei Möglichkeiten zur Verfügung:

- Bundles werden anhand des *Speicherortes*, von dem sie bei der Installation bezogen wurden, authentifiziert.
- Bundles werden anhand ihrer *digitalen Signaturen* authentifiziert.

In der OSGi Spezifikation ist definiert wie Bundles digital signiert werden [OSG09a]. Eine Signatur stellt eine Verbindung zwischen einem Bundle und dem Benutzer, der es signiert hat, her. Aufgrund des Vertrauens zu dem Benutzer können die Berechtigungen für das Bundle vergeben werden.

3 Verwandte Arbeiten

Dieses Kapitel stellt Arbeiten vor, die sich mit der Isolation von Desktop-Grid- oder OSGi-Anwendungen beschäftigen.

3.1 Desktop-Grid-Plattformen

Dieser Abschnitt befasst sich mit Desktop-Grid-Plattformen, die Isolation zur Wahrung der Sicherheit von Anwendungen oder des Wirtsystems einsetzen.

3.1.1 Entropia: Isolation durch binäre Instrumentierung

Entropia [CCWY05] ist die Desktop-Grid-Plattform des *Entropia Desktop Distributed Computing Grid* (DCGrid) [CCEB03], das als Enterprise Desktop Grid konzipiert ist.

Zur Isolation der Anwendungen kommt binäre Instrumentierung zum Einsatz. Vor der Installation einer Anwendung auf den Wirtsystemen wird deren Code modifiziert, so dass Systemaufrufe durch die *Entropia Virtual Machine* abgefangen und überprüft werden können. Da die Instrumentierung auf der Ebene von Binärcode arbeitet, ist das Verfahren unabhängig von der eingesetzten Programmiersprache. Jedoch sind unterschiedliche Betriebssysteme oft nicht binärkompatibel, wodurch für jedes System eine eigene Umsetzung der binären Instrumentierung benötigt wird. Entropia unterstützt nur Windows-32-Bit-Systeme, wodurch die Einsetzbarkeit auf Wirtsysteme mit Betriebssystemen dieser Art beschränkt bleibt.

Damit die binäre Instrumentierung nicht umgangen werden kann, muss den Anwendungen die Ausführung von selbstmodifizierendem Code verweigert werden, da dadurch der Code zum Abfangen der Systemaufrufe wieder entfernt werden kann oder neuer Code ohne Instrumentierung erzeugt werden kann. Moderne Java Virtual Machines setzen jedoch Just-In-Time-Kompilierung ein, die zur Laufzeit Binärcode produziert. Bei der Ausführung von Java-Anwendung muss also die Ausführung von selbtsmodifizierendem Code weiterhin möglich sein, was ein Risiko darstellt, da die Sicherheitsmechanismen von Entropia umgangen werden können.

Zum Schutz des Massenspeichers wird die Nutzung der Datei-Ein- und Ausgabe auf ein, von Entropia kontrolliertes, Verzeichnis umgeleitet. Der Inhalt des Verzeichnisses kann wahlweise

verschlüsselt werden. Die Umleitung und Verschlüsselung geschieht für die Anwendungen transparent. Der Zugriff auf Systemverzeichnisse, die relevante Dateien für die Ausführung der Anwendungen enthalten, wird nicht umgeleitet. Jedoch wird den Anwendungen der Schreibzugriff auf diese Verzeichnisse verweigert. Zugriff auf die Windows-Registry werden auf ähnliche Weise umgeleitet oder auf den Lesezugriff beschränkt.

Den Anwendungen steht nur ein eingeschränkter Netzwerkzugriff zur Verfügung, der auf bestimmte IP-Adressen begrenzt wird, z. B. einen Datenbank-Server o.ä. Im Normalfall darf eine Anwendung keine Netzwerkverbindung herstellen. Stattdessen werden die Berechnungsergebnisse von den Anwendungen in Dateien gespeichert, die von Entropia an einen zentralen Server gesendet werden. Dieses Kommunikationsmodell ist für Irregularly Structured Problems ungeeignet, da durch den zentralen Server ein Engpass entstehen kann.

Des Weiteren werden Fenster der graphischen Benutzungsoberfläche vor dem Benutzer versteckt, so dass er durch die Fenster der Anwendungen nicht gestört werden kann.

Da alle Systemaufrufe abgefangen werden, kann, durch eine Begrenzung der Aufrufe für den Netzwerk- und Datei-Zugriff, die Prozess-/Thread-Erzeugung usw., eine Ressourcenkontrolle umgesetzt werden. Dazu setzt Entropia einen *Desktop Controller* auf dem Wirtsystem ein, der den Ressourcenverbrauch aller Desktop-Grid-Anwendungen überwacht und wenn nötig begrenzt.

Die Authentizität der im Grid verteilten Anwendungen wird durch digitale Signaturen sichergestellt, die auch gleichzeitig als Nachweis für die stattgefundene Instrumentierung dienen.

3.1.2 XtremWeb: Isolation durch Linux Security Modules

In [CDF+05] wird die Isolation von Anwendungen auf Basis von *Linux Security Modules* (LSM) vorgestellt. Diese Funktionalität ist als Erweiterung der Desktop-Grid-Plattform *XtremWeb* gedacht. Durch LSM wird vom Linux-Kernel eine Schnittstelle bereitgestellt, um bei Systemaufrufen eine Berechtigungsprüfung durchführen zu können. Der Kernel ruft dazu das jeweilige Sicherheitsmodul auf, das den Systemaufruf zulässt oder dessen Ausführung verweigert. In [CDF+05] wird SBLSM eingesetzt, ein Modul das speziell für Grid Computing und Peer-to-Peer-Systeme entwickelt wurde. Die Berechtigungsprüfung wird für drei verschiedene Aspekte durchgeführt:

- Datei-Ein/-Ausgabe
- Netzwerkzugriffe
- Interprozesskommunikation (Signale und `ptrace`)

Die Installation und Konfiguration von Sicherheitsmodulen kann den durchschnittlichen Benutzer jedoch überfordern. Zudem ist die Einsetzbarkeit auf Wirtsysteme mit Linux beschränkt.

3.1.3 Isolation durch Systemvirtualisierung

Harmony ist ein Beispiel für eine Desktop-Grid-Plattform, die Systemvirtualisierung zur Isolation der Anwendungen einsetzt [NSBK03]. Eine Anwendung wird dazu in einer virtuellen Maschine ausgeführt, die ein Gastbetriebssystem zur Verfügung stellt. Die Virtualisierung dient dabei nicht nur zur Wahrung der Sicherheit, sondern soll die Anwendungen auch vor den Unterschieden der heterogenen Wirtsysteme abschirmen, um eine höhere Portabilität zu erreichen.

In [MKFL08] wird ein System zur Isolation durch Systemvirtualisierung vorgestellt, das für die Integration mit unterschiedlichen Desktop-Grid-Plattformen konzipiert wurde. Wie in Harmony werden die Anwendungen in virtuellen Maschine ausgeführt. Die Virtualisierung dient dabei zusätzlich als Grundlage für die Umsetzung eines Checkpoint-Mechanismus mit Hilfe dessen die Ausführung beliebiger Anwendungen angehalten werden kann und zum späteren Zeitpunkt fortgesetzt werden kann. Auch die Migration des Zustands auf eine anderes Wirtssystem ist damit möglich.

3.2 Isolation von OSGi-Anwendungen

In [GTFC08] und [GTM+09] werden Ergebnisse von Forschungsarbeiten vorgestellt, die sich mit der Isolation von OSGi-Bundles durch Anpassung der Java Virtual Machine (JVM) befassen. Bisher fehlt in Java das Konzept eines Bundles/Moduls, wodurch, bei der Ausführung mehrerer Bundles in einer JVM, Sicherheitsrisiken entstehen können, z. B. durch die Manipulation von Singleton-Ressourcen. Mit der Einführung einer eigenen Umgebung für Bundles innerhalb einer JVM soll diesen Risiken entgegengewirkt werden. Auch wird die Möglichkeit zur Ressourcenüberwachung einzelner Bundles gegeben, um Angriffe, die auf die Verfügbarkeit von Ressourcen des Wirtsystems abzielen, auf einzelne Bundles zurückführen zu können.

Des Weiteren existieren Ansätze zur Isolation einzelner Services. Dadurch soll die Verwendung von *Stale References* oder fehlerhaften Services verhindert werden [GRD08, AOS06]. Wird die Veröffentlichung eines Services von einem Bundle zurückgezogen, kann der damit verbundene Arbeitsspeicher erst freigegeben werden, wenn alle Konsumenten die Referenz auf das Service-Objekt freigegeben haben. Das liegt an der Speicherverwaltung der JVM. Die Garbage Collection entfernt eine Objekt erst aus dem Speicher, wenn keine Referenzen mehr darauf existieren. Um diesem Problem zu vermeiden, wird das OSGi-Framework modifiziert, so dass für die Konsumenten kein direkter Zugriff mehr auf die Service-Objekte möglich

ist. Dazu werden Proxies eingesetzt, die als transparente Schnittstelle zwischen Service-Konsument und -Anbieter eingefügt werden. Die Referenz auf das Service-Objekt bleibt dabei im Proxy-Objekt verborgen. Wird die Veröffentlichung eines Services zurückgezogen, lässt das Proxy-Objekt die Referenz fallen. Anschließend Aufrufe des Services werden mit einer Exception beantwortet. Da die einzige Referenz im Proxy enthalten war und diese freigegeben wurde, kann auch der zugehörige Speicher freigegeben werden.

In [AOS06] wird zudem vorgestellt, wie die Nutzung von fehlerhaften Services eingedämmt werden kann. Bei der Weiterleitung von Aufrufen durch den Proxy wird zusätzlich das Ergebnis des Aufrufs überprüft. Treten dabei häufig Fehler auf, wird das Service-Objekt, wenn möglich, durch einen anderen Service ersetzt, und das Bundle des fehlerhaften Service entfernt. Es findet also, transparent für den Konsumenten, ein Service-Recovery statt.

3.3 Fazit

Die vorgestellten Ansätze zur Isolation von Desktop-Grid-Anwendungen sind auf bestimmte Arten der Isolation spezialisiert. Die Abwägung zwischen Leistung und Sicherheit kann nur begrenzt stattfinden, indem unterschiedliche Berechtigungen vergeben werden. Unterschiedliche Isolationsarten sind bei keinem der Systeme wählbar, so dass die Art nicht auf den Anwendungsfall abgestimmt werden kann.

Die vorgestellten Arbeiten zur Isolation von OSGi-Anwendungen schlagen eine Anpassung der OSGi-Implementierung oder sogar der Java Virtual Machine vor. Diese Anpassung sind jedoch bisher nicht standardisiert und somit im Produktiveinsatz nicht verfügbar, da nur experimentelle Implementierungen existieren. Das Ziel dieser Arbeit ist es, auf bestehenden Standards aufzubauen, um Kompatibilität mit vorhandener Software zu erreichen.

4 Architektur

Die in diesem Dokument beschriebene Software ergänzt eine Desktop-Grid-Plattform um ein System zur Isolation der Anwendungen. Eine Desktop-Grid-Plattform stellt Anwendungen Services zur Verfügung, die die Kommunikation mit anderen Teilnehmern des Grids ermöglichen und weitere Aufgaben erfüllen, z. B. Lastverteilung. Das System baut auf OSGi auf, ein Framework zur Modularisierung von Software.

Im Systemmodell (4.1) werden die Annahmen erläutert auf deren Grundlage das System entwickelt wurde. Die Komponenten des Systems werden in in Abschnitt 4.2 beschrieben. Die einzelnen Aufgaben des Isolationssystems werden in Abschnitt 4.3 beschrieben. Abschnitt 4.4 befasst sich mit der Kommunikation zwischen der Desktop-Grid-Plattform und den isolierten Anwendungen.

4.1 Systemmodell

Das System dient der Isolation von Anwendungen einer Desktop-Grid-Plattform. Wenn im Folgenden von *Anwendungen* die Rede ist, sind damit die Anwendungen einer Desktop-Grid-Plattform gemeint, wenn nicht anders angegeben. Ein Computer, auf dem eine Desktop-Grid-Plattform und deren Anwendungen ausgeführt wird, wird im Folgenden *Wirtsystem* genannt.

Die Anwendungen werden auf Wirtsystemen ausgeführt, die nicht unbedingt einer zentralen Kontrolle unterliegen. Das bedeutet, dass die Nutznießer der Anwendungen, die von deren Ausführung profitieren, unter Umständen keine Kontrolle über die Betriebsumgebung haben, insbesondere über die gleichzeitig ausgeführten Anwendungen.

Die *Beteiligten* am System werden in folgende Rollen unterschieden.

Nutznießer Ein Nutznießer profitiert von der Ausführung einer Anwendung, z. B. ein Wissenschaftler der die Berechnungsergebnisse einer Anwendung für seine Forschungsarbeit benötigt. Er ist der Urheber der Anwendung oder hat die Entwicklung der Anwendung in Auftrag gegeben.

Benutzer Ein Benutzer stellt einen Computer als Wirtsystem zur Verfügung, nutzt ihn aber auch anderweitig.

Betreiber Der Betreiber verwaltet das Desktop-Grid-System. Er überwacht das Desktop-Grid-System und verteilt neue Anwendungen auf die beteiligten Wirtsysteme. Die Anwendungen werden direkt verteilt, wenn der Betreiber gleichzeitig Benutzer der Wirtsysteme ist, also Zugriff darauf hat, z. B. der Administrator eines Computernetzwerks. Eine weitere Möglichkeit ist es, die Anwendungen, z. B. auf einer Website, zur Verfügung zu stellen, damit Benutzer sie herunterladen können. Der Betreiber tritt also als Vermittler zwischen Nutznießern und Benutzern ein.

Die Desktop-Grid-Plattform nutzt OSGi zur Modularisierung der Anwendungen. Die Anwendungen der Plattform liegen als Bundles vor. Ein Bundle stellt ein Modul in der Terminologie von OSGi dar. Eine Anwendung entspricht genau einem Bundle. Das Bundle kann jedoch Abhängigkeiten zu bestimmten Java-Packages besitzen, die genau spezifiziert sind (mittels des Manifest-Headers `Import-Package`). Dadurch entstehen Abhängigkeiten zu anderen Bundles. Kommunikation mit der Desktop-Grid-Plattform oder anderen Anwendungen findet ausschließlich über OSGi-Services statt. Eine Anwendung ist vornehmlich Konsument von Services und stellt nur in Ausnahmefällen eigene Services bereit.

Der vorrangige Beweggrund für die Isolation von Anwendungen ist die Wahrung der Sicherheit der Anwendungen und des Wirtsystems. Unter Sicherheit wird die Fähigkeit einer Anwendung oder eines Wirtsystems verstanden, die Vertraulichkeit und Integrität der Daten sowie die Verfügbarkeit der Dienste zu gewährleisten. Andere Gründe für die Isolation, wie z. B. das Umgehen von Schwächen in der Architektur einer Anwendung oder des Wirtsystems (siehe 1.2), werden nur nachrangig betrachtet.

Die Isolation einer Anwendung wird mit Hilfe einer virtuellen Umgebung, z. B. einer virtuellen Maschine, bewerkstelligt, in der die Anwendung ausgeführt wird. Es gibt unterschiedliche Arten solcher Umgebungen. Jede Art weist unterschiedliche Isolationseigenschaften auf. Zudem unterliegen die im System verwendeten Isolationsarten einer totalen Ordnung bezüglich des Isolationsgrades. Je höher der Isolationsgrad, desto mehr Interaktionen zwischen einer isolierten Anwendung und dem Wirtsystem oder Anwendungen anderer Umgebungen werden entgegengewirkt.

Mit dem Isolationsgrad steigt auch der zusätzliche Verbrauch an Ressourcen des Wirtsystems, während der Ausführung einer isolierten Anwendung, bedingt durch den Aufwand zur Isolation. Dieser Aufwand fällt für jede Umgebung an, d. h. je höher die Anzahl an Umgebungen, desto höher der zusätzliche Verbrauch an Ressourcen.

Die Desktop-Grid-Plattform kommt vorwiegend auf aktuellen PCs zum Einsatz, die mit zwei oder mehr Prozessorkernen und einer Taktrate von mindestens 2,8 GHz sowie mindestens 2 GB Arbeitsspeicher ausgestattet sind.

4.2 Komponenten

Die zu ergänzende Desktop-Grid-Plattform basiert auf OSGi (siehe 4.1). Das OSGi-Framework, das die Bundles der Desktop-Grid-Plattform enthält, wird im Folgenden *Haupt-Framework* genannt. Im Haupt-Framework werden auch die zu isolierenden Anwendungen verwaltet. Zudem stellt das Haupt-Framework *Kern-Services* bereit. Kern-Services sind OSGI-Services, die allen Anwendungen, unabhängig von deren Isolationsgrad, zur Verfügung stehen.

4.2.1 Isolationsstufen und -umgebungen

Eine isolierte Anwendung wird in genau einer *Isolationsumgebung* ausgeführt. Es können jedoch mehrere Anwendungen in derselben Umgebung ausgeführt werden.

Eine Isolationsumgebung gehört einer bestimmten *Isolationsstufe* an und wird vom System erzeugt und zerstört. Eine Isolationsstufe repräsentiert eine bestimmte Isolationsart (siehe 4.1). Wenn im Folgenden davon die Rede ist, dass eine Anwendung in einer bestimmten Isolationsstufe ausgeführt wird, dann ist damit die Ausführung der Anwendung in einer Isolationsumgebung der jeweiligen Stufe gemeint.

Es kann beliebig viele Isolationsumgebungen einer bestimmten Isolationsstufe geben mit Ausnahme der ersten Isolationsstufe. Die erste Stufe repräsentiert immer die Stufe *Keine Isolation*. Es gibt immer genau eine Isolationsumgebung dieser Stufe: der Prozess, in dem die Desktop-Grid-Plattform ausgeführt wird.

Die *Stärke* einer Isolationsstufe bezeichnet den Isolationsgrad (siehe 4.1), den die zugehörigen Isolationsumgebungen bieten.

Als *schwächste Isolationsstufe* wird die Stufe mit dem niedrigsten Isolationsgrad bezeichnet, die stärker isoliert als die Stufe *Keine Isolation*.

In Kapitel 5 werden konkrete Isolationsstufen beschrieben.

4.2.2 Haupt-Framework

Das *Haupt-Framework* ist der Prozess, der das OSGi-Framework ausführt, das die Desktop-Grid-Plattform enthält. Im Haupt-Framework wird auch das Isolationssystem ausgeführt. Gleichzeitig stellt das Haupt-Framework die einzige Isolationsumgebung der Isolationsstufe *Keine Isolation* dar. Anwendungen, die im Haupt-Framework ausgeführt werden, werden als nicht isoliert angesehen.

Das Haupt-Framework stellt die Services der Desktop-Grid-Plattform auch anderen Isolationsumgebungen zur Verfügung. Services die nicht nur lokal in einem Framework bereitgestellt werden, werden im Folgenden *Kern-Services* genannt.

4.2.3 Kind- und Satellit-Frameworks

Da es sich bei den Anwendungen um Bundles handelt, wird für deren Ausführung ein OSGi-Framework benötigt. Eine Isolationsumgebung enthält genau ein sogenanntes *Kind-Framework*, das die zu isolierenden Anwendung aufnimmt. Die Anwendungen eines Kind-Frameworks können Anwendungen außerhalb des Frameworks weder Packages noch Services bereitstellen.

Ein *Satellit-Framework* ist ein Kind-Framework, das Kern-Services zur Verfügung stellen kann.

4.3 Aufgaben des Isolationssystems

Die Aufgaben des Isolationssystems lassen sich auf vier Teilgebiete aufteilen:

- Auflösung von Abhängigkeiten
- Auflösung von Isolationsvorgaben
- Verwaltung der Isolationsumgebungen/Anwendungen
- Gewährleisten der Kommunikation

Eine Anwendung besteht aus einem Bundle, das Abhängigkeiten zu anderen Bundles haben kann (siehe 4.1). Diese Abhängigkeiten müssen vor der Installation des Systems aufgelöst werden, also die benötigten Bundles beschafft werden. Dazu kann z. B. ein *OSGi Bundle Repository* verwendet werden [OSG09c]. Dabei handelt es sich um ein Archiv von Bundles. Anhand der Metadaten eines Bundles lassen sich dessen Abhängigkeiten feststellen und das Archiv kann danach durchsucht werden.

Liegen alle Bundles der Anwendung vor, muss vor deren Ausführung bestimmt werden, in welcher Isolationsumgebung die Anwendung ausgeführt werden soll. Dazu werden Vorgaben ausgewertet, die von den Beteiligten des Systems gestellt werden. Wie die Auflösung der Isolationsvorgaben umgesetzt werden kann, ist in Kapitel 6 beschrieben.

Nachdem die Isolationsumgebungen der Anwendungen feststehen, werden sie erzeugt und die Bundles der Anwendungen in den Kind-Frameworks ausgeführt.

Damit die Anwendungen trotz der Isolation noch die Dienste der Desktop-Grid-Plattform in Anspruch nehmen können, muss die Kommunikation zwischen den isolierten Anwendungen

und der Plattform gewährleistet werden. Die Umsetzung dieser Anforderung ist im nächsten Abschnitt beschrieben.

4.4 Kommunikation

Trotz Isolation muss zwischen den Isolationsumgebungen die Möglichkeit zur Kommunikation bestehen. Denn isolierte Anwendungen müssen auf die Funktionalität der Desktop-Grid-Plattform zugreifen können. Zudem soll es möglich sein, dass Anwendungen die Plattform um Funktionalität erweitern. Dazu kann z. B. das Whiteboard-Pattern eingesetzt werden: Eine Anwendung registriert einen Service eines bestimmten Typs (eines bestimmten Java-Interfaces). Die Plattform hält nach Services diesen Typs Ausschau und nutzt deren Funktionalität, wenn sie verfügbar sind. Damit dieses Entwurfsmuster umgesetzt werden kann, müssen auch isolierte Anwendungen Kern-Services bereitstellen können.

Die Aufgabe besteht also darin, die Services eines Frameworks auch Bundles in anderen Frameworks zur Verfügung zu stellen.

4.4.1 OSGi Remote Services

Um Services Framework-übergreifend zur Verfügung stellen zu können, werden *Remote Services* eingesetzt. Diese werden im Service Compendium der OSGi-Spezifikation definiert [OSG09b]. Ein *Distribution Provider* stellt anderen Frameworks einen lokalen Service zur Verfügung. Er *exportiert* einen Service. Zudem ist er dafür verantwortlich Remote Services zu *importieren*, also im lokalen Framework verfügbar zu machen. Dazu wird ein Proxy-Service im Framework registriert, der die transparente Nutzung des Remote Service ermöglicht. Für OSGi Services gilt *Location Transparency*, d. h. der Ort des Services ist dem Benutzer nicht bekannt.

Die Umsetzung des Distribution Providers wird in der Spezifikation nicht beschrieben. Es wird lediglich definiert, wie dem Distribution Provider mitgeteilt wird, dass ein bestimmter Service exportiert werden soll. Dazu wird den Metadaten des zu exportierenden Services ein bestimmtes Property hinzugefügt. Es gibt die Java-Interfaces an, unter denen der Service in anderen Frameworks zur Verfügung gestellt werden soll. Wie ein Distribution Provider einen Remote Service auffindet, bleibt ebenfalls der Implementierung überlassen, da hierzu keine Angaben in der Spezifikation gemacht werden.

Im Kontext von OSGi Remote Services ist die Transparenz bei der Nutzung der Services als vorrangig zu betrachten, da auch eine möglichst weitreichende Unterstützung für Anwendungen geboten werden soll, die nicht im Kontext einer Desktop-Grid-Plattform, die ein Isolationssystem einsetzt, entwickelt wurden.

4.4.2 Implementierungen von OSGi Remote Services

Bisher existieren zwei Implementierungen von standardkonformen Distribution Providern:

- Apache CXF (Referenzimplementierung) [Apaa]
- Eclipse Communication Framework [Ecla]

Apache CXF ist eine Java-Programmbibliothek, die Funktionalität zur Bereitstellung und Nutzung von Web-Services enthält. Die Unterstützung für OSGi Remote Services wird als Unterprojekt entwickelt. Das *Eclipse Communication Framework* (ECF) unterstützt den Entwickler bei der Erstellung von verteilten Anwendungen auf Basis der *Eclipse Rich Client Platform* [Eclb].

Beide Programmbibliotheken bieten verschiedene Paradigmen und Protokolle zur Umsetzung der Kommunikation zwischen entfernten Anwendungen. Diese stehen auch bei der Nutzung der Distribution Provider zur Verfügung.

Bei den Implementierungen ist auf die Einhaltung von Standards zu achten, so dass keine Abhängigkeit zu einer bestimmten Implementierung entsteht und damit die Austauschbarkeit der jeweiligen Komponente gefährdet wird. Im Fall von ECF 3.2 bestehen Abhängigkeiten zu Equinox, der OSGi-Implementierung der Eclipse Foundation.

Die Implementierung von Apache CXF hingegen kann unabhängig von der OSGi-Implementierung genutzt werden.

In beiden Implementierungen ist die Funktionalität zum Import von Remote Services auf das Auffinden der Services im lokalen Netzwerk oder dem Internet ausgelegt. Ein solche Funktionalität wird für das Isolationssystem nicht benötigt. Zudem entstehen dadurch unnötige Abhängigkeiten für das System.

Aus den oben genannten Gründen, wurde entschieden eine eigene Implementierung eines Distribution Providers zu entwickeln, der auf die Anforderungen des Isolationssystems zugeschnitten ist. Die Implementierung ist im nächsten Abschnitt beschrieben.

4.4.3 Implementierung auf Basis von R-OSGi

Als Grundlage wurde *R-OSGi* 1.0.0.RC4 verwendet [RAR07]. Dabei handelt es sich um einen Distribution Provider der bereits vor Fertigstellung der Remote-Services-Spezifikation entwickelt wurde. Deshalb ist er nicht standardkonform. Zur Kommunikation verwendet R-OSGi ein eigenes binäres Protokoll, das die auf TCP aufbaut. R-OSGi ist jedoch auch erweiterbar für andere Protokolle. Es existiert ein Erweiterung, die auf Apache MINA [Apab] aufbaut, sowie ein Protokoll zur Bluetooth-Kommunikation. Für das Isolationssystem wurde das R-OSGi-eigene Protokoll verwendet.

R-OSGi bietet keine Transparenz bei der Anforderung von Remote Services. Es muss ein spezieller Service genutzt werden, um Remote Services in das lokale Framework zu importieren. Erst nachdem Import ist die Location Transparency hergestellt, wodurch der Service auch über die Dienste des Frameworks angefordert werden kann.

Deshalb wurde ein Bundle implementiert, dass den Import eines Remote Services anstößt, sobald der Service von einem Bundle im Framework angefragt wird. Beim Start einer Isolationsumgebung wird deren Kind-Framework zunächst über R-OSGi mit dem Haupt-Framework und den Satelliten-Frameworks verbunden. Anschließend werden die Anfragen im lokalen Framework überwacht. Falls eine Service angefragt wird, der als Kern-Service von einem anderen Framework zur Verfügung gestellt wird, wird er mittels R-OSGi importiert.

Der Export eines Service wird bei R-OSGi, genauso wie bei den OSGi Remote Services, durch ein Property des zu exportierenden Services ausgelöst. Allerdings wird von R-OSGi ein Property mit einem anderen Namen definiert als in der OSGi-Spezifikation. Jedoch bietet R-OSGi eine weitere Möglichkeit Services für den Export zu markieren. Es kann ein Stellvertreter-Service im lokalen Framework registriert werden, der auf den zu exportierenden Service verweist. Dadurch ist es auch möglich Services, die kein bestimmtes Property definieren, zu exportieren. Das ist z. B. bei Anwendungen hilfreich, die nicht für die Nutzung mit Remote Services ausgelegt wurden.

Das implementierte Bundle registriert also Stellvertreter-Services für die zu exportierenden Kern-Services einer Anwendung.

5 Isolationsstufen

Eine Isolationsstufe beschreibt einen bestimmten Typ von Isolationsumgebungen. Im Folgenden werden einige konkrete Isolationsstufen vorgestellt.

5.1 Keine Isolation

Die erste Isolationsstufe ist immer die Stufe *Keine Isolation*. Sie definiert den nicht-isolierten Zustand. Im Gegensatz zu den anderen Isolationsstufen besitzt sie nur genau eine Isolationsumgebung: das Haupt-Framework. Eine Anwendung, die in dieser Umgebung ausgeführt wird, läuft Seite-an-Seite mit der Desktop-Grid-Plattform in einem OSGi-Framework. Die Isolation zwischen Anwendungen und der Desktop-Grid-Plattform findet nur auf Basis von OSGi statt.

5.1.1 Isolation in OSGi-Frameworks

Die Einheiten, die in einem OSGi-Framework voneinander isoliert werden, sind Bundles, die Module von OSGi. Da Anwendungen aus Bundles bestehen, kommt dies der Anwendungsisolation gleich.

In den Abschnitten 2.2.1 und 2.2.2 wurde bereits beschrieben wie sich die Isolation auswirkt: Java-Pakete eines Bundles sind nur nach expliziter Veröffentlichung für andere Bundles sichtbar. Java-Objekte sind, in Form von OSGi-Services, ebenfalls nur nach expliziter Veröffentlichung für andere Bundles sichtbar. Zudem kann die Nutzung von Services und Packages durch das Entziehen von Berechtigungen eingeschränkt werden. Es findet also eine Isolation von Java-Paketen und -Objekten statt.

5.1.1.1 Class Loader

Umgesetzt wird diese Isolation mit Hilfe der *Class Loader* von Java. Class Loader sind dafür verantwortlich den Java Code der Klassen in den Arbeitsspeicher zu laden, so dass er ausgeführt werden kann [LB98]. Dazu werden Klassendateien (engl. class file) eingelesen und der Java Virtual Machine zur Ausführung zur Verfügung gestellt. Class Loader sind gewöhnliche Java-Objekte. Es können also auch anwendungsspezifische Class Loader in Java

geschrieben werden, um Klassen nicht nur von der Festplatte oder aus JAR-Dateien zu laden, sondern z. B. auch über eine Netzwerkverbindung von einem anderen Computer.

Um anwendungsspezifische Class Loader nutzen zu können, müssen diese natürlich auch erst geladen werden. Deshalb werden von der Java Laufzeitumgebung bereits einige vordefinierte Class Loader bereitgestellt.

Das Laden der ersten Klassen eines Programms wird vom *Bootstrap Class Loader* durchgeführt. Er ist dafür verantwortlich die Systemklassen zu laden. Das sind z. B. die Klassen aus den `java.*`-Paketen, wie `java.lang.String`. Der *Extension Class Loader* kann optionale Java-Pakete zur Erweiterung der Laufzeitumgebung laden [GE03, Sun06a]. Die anwendungsspezifischen Klassen eines Java-Programms werden vom *Application Class Loader* [1] geladen [GE03].

5.1.1.2 Delegieren des Class Loading

Enthält eine Klasse C eine Referenz auf eine noch nicht geladene Klasse D, wird der Class Loader L_C, der C geladen hat, aufgefordert, D zu laden. Dieser kann selbst die Klasse D *definieren*, also der Java Virtual Machine zur Verfügung stellen. Ein Class Loader kann auch an einen anderen Class Loader *delegieren*, der die Klasse definieren soll. Das Laden von Systemklassen z. B. wird letztendlich immer an den Bootstrap Class Loader delegiert, der alle Systemklassen definiert. Abbildung 5.1 zeigt ein Beispiel, in dem der Class Loader von L_C das Laden der Klasse D an den Class Loader L_D delegiert.

Der Bezeichner einer Java-Klasse setzt sich aus dem Namen des Pakets, in dem die Klasse enthalten ist, und dem Klassennamen zusammen. Ein Paket bildet also einen Namensraum für die enthaltenen Klassen, so dass kein Konflikt entsteht, wenn zwei Klassen mit identischen Namen existieren, sie sich aber in unterschiedlichen Paketen befinden. Zur Laufzeit wird eine Klasse, zusätzlich zu ihrem Bezeichner, noch durch den Class Loader identifiziert, der sie geladen hat. Mit Hilfe eines Class Loader lässt sich also ein eigener Namensraum aufspannen: In einer Java Virtual Machine können zwei unterschiedliche Klassen mit gleichem Bezeichner existieren, wenn sie durch unterschiedliche Class Loader geladen wurden. Durch die Class Loader lassen sich also Namensräume voneinander isolieren. Im Beispiel der Abbildung 5.1 existieren zwei Definitionen der Klasse D, die durch unterschiedliche Class Loader geladen wurden.

[1] Während der Entwicklung von Java 2 gab es einige Änderungen bei der Benennung der Class Loader [GE03]. Der Bootstrap Class Loader wurde vorher auch *System Class Loader* genannt [LB98]. In Java 2 ist der System Class Loader jedoch für das Laden der anwendungsspezifischen Klassen verantwortlich. Um Verwirrung zu vermeiden, wird in diesem Dokument die Terminologie aus [GE03] verwendet. Dabei wird die neue Definition des Begriffs *System Class Loader* vermieden und stattdessen der Begriff *Application Class Loader* verwendet.

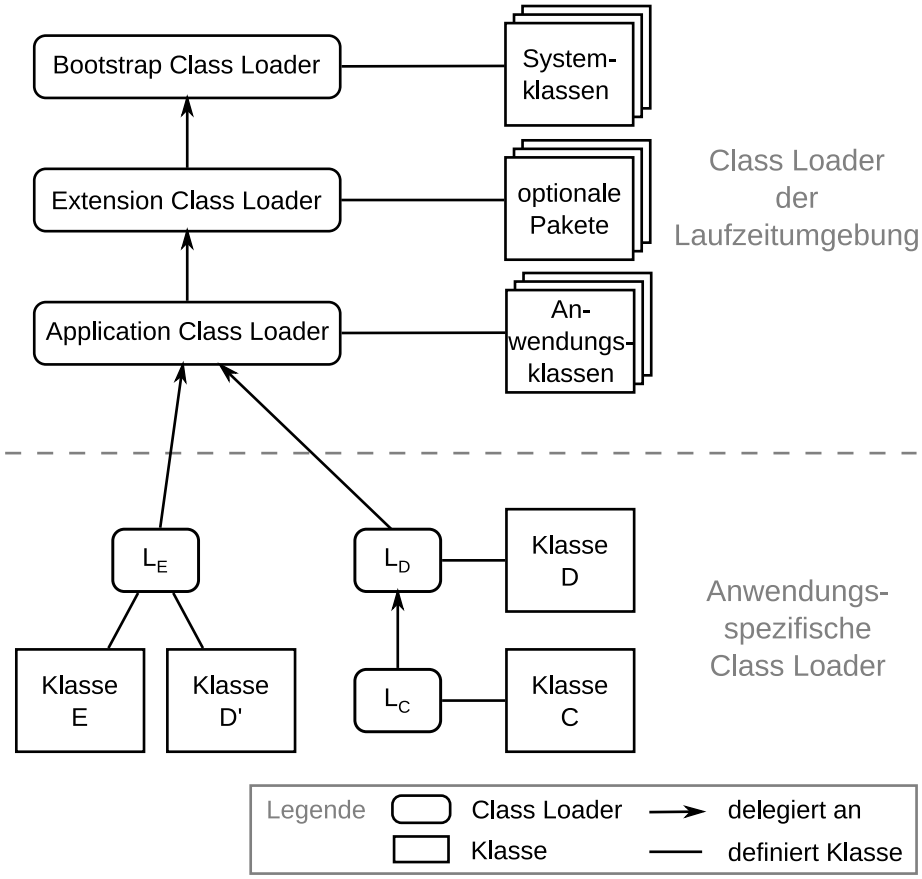

Abbildung 5.1: Delegieren des Class Loading

5.1.1.3 Class Loader in OSGi

Ein OSGi-Framework definiert für jedes Bundle einen eigenen Class Loader. Dadurch entsteht für jedes Bundle ein eigener Namensraum. Dieser muss beim Export von Java-Paketen jedoch teilweise für andere Bundles geöffnet werden. Soll ein Paket von einem Bundle importiert werden, delegiert der Class Loader des importierenden Bundles alle Anfragen an Klassen aus dem jeweiligen Paket an den Class Loader, der das Paket exportiert. Der Class Loader des exportierenden Bundles macht also die exportierten Java-Pakete aus seinem eigenen Namensraum anderen Bundles zugänglich.

Im Gegensatz zu dem streng hierarchischen Delegationsgraphen, der im Beispiel der Java Class Loader (Abb. 5.1) vorgestellt wurde, bilden die Class Loader in OSGi einen Graph, das sich im Allgemeinen nicht als Baum anordnen lässt. Abbildung 5.2 zeigt ein Beispiel für ein Class-Loader-Netzwerk, das von einem OSGi-Framework aufgespannt werden kann. Systemklassen (z. B. `java.*`) werden immer vom *Parent Class Loader* geladen. Im Beispiel ist

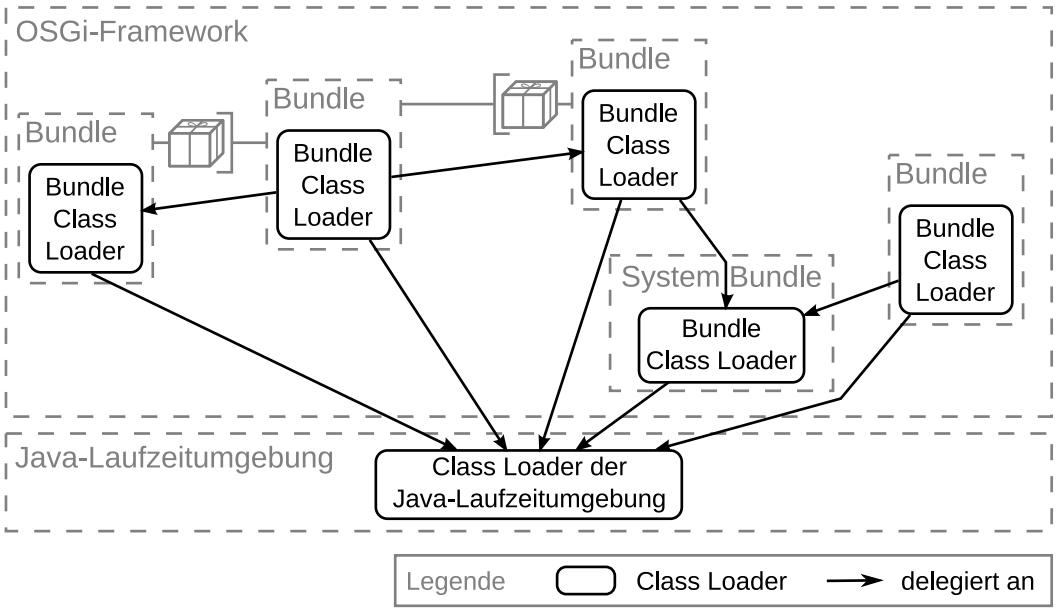

Abbildung 5.2: Class Loader in OSGi

dies ein Class Loader der Laufzeitumgebung. Es ist jedoch auch möglich, dass der Parent Class Loader vom OSGi-Framework implementiert wird und das Laden der Systemklassen an einen Class Loader der Laufzeitumgebung delegiert. Das *System Bundle* ist ein spezielles Bundle, das vom Framework zur Verfügung gestellt wird. Es exportiert Java-Pakete der OSGi-Implementierung, die den Bundles zur Verfügung stehen müssen, um Dienste des Frameworks nutzen zu können, wie z. B. das Veröffentlichen von Services. Die *Bundle Class Loader* laden die, im zugehörigen Bundle enthaltenen, Pakete und delegieren das Laden von importierten Paketen an den Class Loader des jeweiligen exportierenden Bundles.

5.1.2 Isolation durch Java-Permissions

Die Java-Laufzeitumgebung führt Programme in einer virtuellen Maschine aus. Die Java Virtual Machine (JVM) überprüft den Programm-Code bevor er tatsächlich ausgeführt wird. Für den Zugriff auf sicherheitsrelevante Dienste können Berechtigungen vergeben werden. Diese Java-Permissions dienen der JVM als Grundlage für die Entscheidung sicherheitsrelevante Zugriffe zuzulassen oder zu verweigern. Durch `FilePermissions` lassen sich z. B. Zugriffe auf Dateien und Verzeichnisse steuern.

OSGi spezifiziert eigene Java-Permissions, die eine feingranulare Berechtigungsvergabe für die Dienste eines Frameworks erlauben (siehe auch 2.2.4). Es kann z. B. festgelegt werden, ob ein Bundle auf Verwaltungsoperationen, wie die Installation oder Deinstallation anderer Bundles, zugreifen darf. Dadurch können Interaktionen zwischen Anwendungen vermieden

werden. Java-Permissions können also verwendet werden, um Anwendungen zusätzlich zu isolieren. Durch die Wahl der eingesetzten Berechtigungen lässt sich der Grad der Isolation an den jeweiligen Anwendungsfall anpassen.

5.2 Mehrere Frameworks in einer Java Virtual Machine

Normalerweise wird in einer JVM nur ein Framework ausgeführt, das die auszuführenden Anwendungen enthält. Im OSGi RFC 138 [OSG09d] wird die Nutzung mehrerer OSGi-Frameworks in einer Java Virtual Machine (JVM) untersucht. Das RFC war Teil des Entwurfs der OSGi-Spezifikation 4.2, wurde jedoch nicht in den Standard übernommen.

Es werden drei Möglichkeiten definiert mehrere Frameworks in einer JVM auszuführen, die im Folgenden beschrieben werden.

5.2.1 Peer Embedded Frameworks

Dieser Fall beschreibt die Ausführung mehrere gleichgestellter Frameworks in einer JVM, die jeweils als Basis unterschiedlicher Anwendungen genutzt werden. Eine Anwendung kann nicht auf Services oder Java-Pakete einer Anwendung aus einem anderen Framework zugreifen. Es ist also keine direkte Interaktion zwischen Anwendungen aus unterschiedlichen Frameworks möglich.

5.2.2 Nested Frameworks

Ein *Nested Framework* wird von einem Bundle verwaltet, das innerhalb eines anderen Frameworks ausgeführt wird. Das Nested Framework ist für das Framework des Bundles nicht sichtbar. Es gilt, wie beim Fall der Peer Embedded Frameworks, dass keine Interaktion zwischen Anwendungen aus den unterschiedlichen Frameworks stattfinden kann. Die Lebensdauer des Nested Framework ist jedoch abhängig von der Lebensdauer des Frameworks in dem das zugehörige Bundle ausgeführt wird.

5.2.3 Child Frameworks

Ein *Child Framework* wird, wie ein Nested Framework, von einem Bundle verwaltet, das innerhalb eines Vater-Frameworks ausgeführt wird. Im Gegensatz zu Nested Frameworks besteht zwischen dem Child Framework und dem Vater-Framework eine direkte Beziehung. Das Child Framework wird, von dem zugehörigen Bundle, als eine neue Instanz des Vater-

Frameworks erzeugt. Zudem können Java-Pakete aus dem Child-Framework dem Vater-Framework zur Verfügung gestellt werden und umgekehrt. Auch Services können ausgetauscht werden.

In einem Child Framework können die Bundles einer Anwendung zusammengefasst werden. Die Anwendung wird im Vater-Framework durch ein sogenanntes *Composite Bundle* repräsentiert. Dieses Bundle veröffentlicht nur die Java-Pakete und Services, die für andere Anwendungen oder die Desktop-Grid-Plattform relevant sind. Services, die nur für die anwendungs-interne Kommunikation gedacht sind, werden vom Child-Framework versteckt.

5.2.4 Singletons

Eine Schwierigkeit bei der Umsetzung aller beschriebenen Möglichkeiten stellen *Singleton-Ressourcen* dar. Solche Singletons[2], sind nicht für den Gebrauch von mehreren unabhängigen Programmen ausgelegt. Sie sind also auch nicht für die Verwendung von mehreren Frameworks bestimmt. Ein Beispiel ist die URL-Klasse von Java, genauer deren Erweiterungsmöglichkeiten. Ein *Uniform Resource Locator* (URL) ist eine Zeichenkette, die den Ort und Zugang zu einer Ressource spezifiziert [BLMM94]. Objekte der URL-Klasse können in Java verwendet werden, um auf die spezifizierten Ressourcen zuzugreifen. Außerdem kann die URL-Klasse erweitert werden, so dass auch auf anwendungsspezifische Ressourcen zugegriffen werden kann. Um der Java-Laufzeitumgebung eine solche Erweiterung hinzuzufügen, wird eine Methode verwendet, die nur einmal aufgerufen werden darf[3]. Folglich kann sie, ohne weitere Vorkehrungen, nicht von mehreren Anwendungen genutzt werden.

Die Singleton-Ressourcen müssen aber jedem der unabhängigen Frameworks zur Verfügung stehen. Es wird also eine Art Multiplexschaltung für jedes Singleton benötigt. Da in einem Framework alleine bereits mehrere Anwendungen ausgeführt werden können, definiert OSGi einen *URL Handlers Service*. Der Service führt eine Schicht zwischen dem Singleton, z. B. der URL-Klasse, und den Anwendungen ein. Diese Schicht ermöglicht die gleichzeitige Verwendung mehrerer Erweiterungen der URL-Klasse. Ein solche Schicht muss auf die ganze JVM ausgedehnt werden, so dass die zugrunde liegenden Singletons unter allen Frameworks und deren Anwendungen aufgeteilt werden und nicht nur innerhalb eines Frameworks. Im OSGi RFC 138 sind Vorschläge für die Umsetzung einiger Multiplex-Schichten enthalten.

Die Anwendung müssen jedoch speziell für die Verwendung dieser Schichten ausgelegt sein. Ein direkter Zugriff auf die Singletons darf nicht mehr erfolgen. Das erfordert meist einen Eingriff in den Quelltext der Anwendungen, was mit hohem Aufwand verbunden sein kann.

[2]Eine Singleton-Ressource entspricht nicht immer einem Klassenobjekt. Deshalb entspricht ein Singleton in diesem Kontext nicht unbedingt dem gleichnamigen Entwurfsmuster aus der Objekt-orientierten Programmierung. Dieses Muster legt fest, dass von einer Klasse nur genau ein Objekt erzeugt werden kann.

[3]Dabei handelt es sich um die Methode `java.net.URL.setURLStreamHandlerFactory` [Suno6b]

5.3 Mehrere Programme in einer Java-Laufzeitumgebung

Im Java Specification Request 121 (JSR 121) wird eine Schnittstelle zur Isolation von Java-Programmen definiert [Suno6c]. Über die Schnittstelle lassen sich sogenannte *Isolates* erzeugen, die Programme aufnehmen. Die Java-Objekte aus einem anderen Isolate sind für ein Programm nicht sichtbar. Auch Singleton-Ressourcen, wie z. B. statische Variablen, sollen keine Möglichkeit zur Interaktion zwischen Isolates bieten. Als Kommunikation werden lediglich einfache Byte-Ströme definiert, die als Verbindung zwischen Isolates aufgebaut werden können.

Um einer Desktop-Grid-Anwendung die nötige Infrastruktur zur Verfügung zu stellen, wird ein OSGi-Framework vorausgesetzt. Eine Anwendung, die mit Hilfe eines Isolate isoliert werden soll, wird also in einem OSGi-Framework ausgeführt, das sich in einem Isolate befindet. Durch die Installation weiterer Anwendungen im Framework, können auch mehrere Anwendungen im Isolate ausgeführt werden.

Wie die Isolation umgesetzt werden soll, ist im JSR 121 nicht spezifiziert. Deshalb können keine genauen Aussagen zu den Isolationseigenschaften getroffen werden.

In der experimentellen Multi-Tasking-Virtual-Machine (MVM) [CD01] wurde der JSR zum Beispiel für die Ausführung mehrerer Programme in einem Prozess umgesetzt. Java-Programme sind in der MVM voneinander isoliert, teilen sich aber eine gemeinsame Java-Laufzeitumgebung. Dadurch wird Arbeitsspeicher eingespart, gegenüber der getrennten Ausführung der Programme in eigenen Laufzeitumgebungen. Native Programmbibliotheken, die von einem Java-Programm geladen werden, werden von der MVM in eigenen Prozessen ausgeführt. Tritt durch eine native Programmbibliothek eine Schutzverletzung o.ä. auf, die eine vorzeitige Beendigung des Prozesses zur Folge hat, wird nur der Prozess der nativen Bibliothek beendet und nicht die gesamte Laufzeitumgebung.

Bisher wurde die Schnittstelle aus dem JSR 121 nicht in den offiziellen Java-API-Standard aufgenommen. Die Umsetzung beschränkt sich auf experimentelle Implementierungen, wie die MVM.

5.4 Anwendung in eigenem Prozess

Betriebssystemprozesse stellen eine weitere Möglichkeit zur Isolation von Anwendungen dar. In einem Prozess wird eine Java-Laufzeitumgebung gestartet, die ein OSGi-Framework ausführt. Das Framework enthält eine oder mehrere zu isolierende Anwendungen. Anwendungen, die in unterschiedlichen Prozessen ausgeführt werden, genießen dadurch die, durch das Betriebssystem gewährleistete, Prozessisolation.

Die Eigenschaften der Prozessisolation sind abhängig vom Betriebssystem, jedoch lassen sich einige allgemeine Aussagen treffen. Die Lebensdauer von Prozessen ist unabhängig. Ein Absturz einer isolierten Anwendung beendet also nicht die Anwendungen aus anderen Prozessen. Prozesse besitzen private Adressräume. Eine isolierte Anwendung besitzt folglich einen geschützten Bereich des Arbeitsspeichers der für prozessfremde Anwendungen nicht einsehbar ist.

5.4.1 Implementierung

Die Isolationsstufe *Prozess* kann in Java mit Hilfe der Klasse `java.lang.ProcessBuilder` umgesetzt werden. Um eine Isolationsumgebung zu erstellen, wird mit dieser Klasse eine Java Virtual Machine (JVM) in einem neuen Prozess gestartet. Die JVM führt den *Framework Launcher* aus. Dabei handelt es sich um ein Programm, das ein OSGi-Framework startet, sowie eine Komponente zur Fernsteuerung des Frameworks, den *Remote Controller*. Er wird zur Verwaltung des Kind-Frameworks verwendet, z. B. zur Installation von Bundles. Zur Kommunikation zwischen dem *Remote Controller* und dem Haupt-Framework wird Remote Method Invocation (RMI) eingesetzt. Nach dem Start des Kind-Frameworks werden zunächst die Bundles installiert, die für die Kommunikation über Remote Services benötigt werden. Zudem wird konfiguriert, welche Kern-Services importiert werden sollen. Die Implementierung der Kommunikation über Remote Services ist in Abschnitt 4.4.3 beschrieben. Anschließend werden die Bundles der Anwendungen installiert und gestartet.

5.5 Systemvirtualisierung

Unter Systemvirtualisierung werden in dieser Arbeit Verfahren verstanden, die auf Software-Ebene die physischen Ressourcen, die Hardware, eines Computersystems aufteilen, so dass auf Basis eines physischen Computersystems mehrere virtuelle Computersysteme bereitgestellt werden können. Ein solches virtuelles Computersystem wird *virtuelle Maschine* (engl. virtual machine) genannt [VNE+08]. Die Software, die die virtuelle Maschine zur Verfügung stellt, wird als *Virtual Machine Monitor* (VMM) bezeichnet [VNE+08]. Da eine virtuelle Maschine nur (virtuelle) Hardware bereitstellt, muss in ihr ein Betriebssystem installiert werden, damit auch Programme auf dem virtuellen Computersystem ausgeführt werden können.

Eine Desktop-Grid-Anwendung benötigt zur Ausführung ein OSGi-Framework. OSGi basiert wiederum auf einer Java-Laufzeitumgebung. Die Java-Laufzeitumgebung wird im Betriebssystem der virtuellen Maschine ausgeführt.

5.6 Evaluierung der Effektivität

Um die Effektivität einer Isolationsstufe festzustellen, muss untersucht werden, welche Interaktionen durch die Isolation unterbunden werden. In dieser Arbeit wird die Sicherheit der Anwendungen und des Wirtsystems als vorrangiger Grund für die Isolation betrachtet (siehe 4.1). Deshalb liegt der Fokus auf Möglichkeiten zur Interaktion, die die Sicherheit gefährden, im Folgenden *Schwachstellen* genannt. Eine Schwachstelle kann *ausgenutzt* werden, d. h. die zugehörige Möglichkeit zur Interaktion wird wahrgenommen. Das Ausnutzen einer oder mehrerer Schwachstellen wird auch als *Angriff* bezeichnet. Das Subjekt eines Angriffs ist der *Angreifer*. Im Folgenden wird unter einem Angreifer eine Anwendung verstanden, die eine Schwachstelle ausnutzt. Kann eine Schwachstelle durch die Isolation nicht mehr ausgenutzt werden, gilt sie als geschlossen. Es wird also untersucht, welche Schwachstellen durch die Isolation geschlossen werden können.

Eine großer Teil der untersuchten Schwachstellen stammt aus einer Arbeit von P. Parrend und S. Frénot. Dabei handelt es sich um einen Katalog von Schwachstellen, der insbesondere OSGi-Frameworks berücksichtigt [PF07]. Zudem haben die Autoren Bundles implementiert, die die Schwachstellen ausnutzen. Die Bundles wurden dem Autor dieser Diplomarbeit freundlicherweise für die Evaluierung zur Verfügung gestellt. Da eine Anwendung im Isolationssystem aus einem oder mehreren Bundles besteht (siehe 4.1), kann der Begriff *Angreifer* für Bundles und Anwendungen gleichbedeutend verwendet werden.

In den folgenden Abschnitten werden die untersuchten Schwachstelle beschrieben. Jede Schwachstelle besitzt einen beschreibenden *Namen*, der in der Abschnittsüberschrift enthalten ist. Außerdem wird für jede Schwachstelle ein eindeutiger Name, eine *ID*, vergeben. Stammt die Schwachstelle aus dem Katalog [PF07] wird die, im Katalog angegebene, ID verwendet. Ansonsten wird eine eigene ID vergeben.

Nach der Überschrift folgt die Beschreibung des Angriffs, der ausgeführt wird, um die Schwachstelle ausnutzen. Zudem ist die Art der Auswirkung des Angriffs angegeben. In [PF07] wird zwischen drei *Angriffsarten* unterschieden: *Nichtverfügbarkeit* (engl. Unavailability), *Leistungseinbruch* (engl. Performance Breakdown) und *unberechtigter Zugriff* (engl. Undue Access). Nichtverfügbarkeit liegt vor, wenn z. B. die Dienste des OSGi-Frameworks blockiert werden oder das Framework beendet wird. Leistungseinbruch bedeutet, dass durch einen Angriff die Ressourcen des Systems, z. B. Prozessor oder Arbeitsspeicher, stark ausgelastet werden. Ein unberechtigter Zugriff beschreibt das Lesen oder Verändern von Daten, die für den Angreifer nicht zugänglich sein sollten, z. B. der Zugriff auf Java-Pakete eines anderen Bundles, die nicht exportiert wurden. In Tabelle 5.1 wird der Zusammenhang zwischen den Angriffsarten und den Sicherheitsprinzipien Vertraulichkeit, Integrität und Verfügbarkeit aufgezeigt.

Art der Auswirkung	gefährdetes Sicherheitsprinzip
unberechtigter Zugriff	Vertraulichkeit, Integrität
Nichtverfügbarkeit	Verfügbarkeit
Leistungseinbruch	Verfügbarkeit

Tabelle 5.1: Zusammenhang zwischen Angriffsarten und Sicherheitsprinzipien der CIA-Triade

Für jede Isolationsstufe wird beschrieben, welche *Auswirkungen* das Ausnutzen der Schwachstelle hat. Die Schwachstelle wird dabei von einer Anwendung ausgenutzt, die sich in einer Isolationsumgebung der jeweiligen Stufe befindet.

Der *Status* erläutert, ob die Schwachstelle durch die jeweilige Isolationsstufe geschlossen wurde oder nicht. Es wird zwischen verschiedenen Statustypen unterschieden:

Schwachstelle offen: Die Schwachstelle kann ausgenutzt werden.

Schwachstelle geschlossen: Die Schwachstelle kann von einer isolierten Anwendung gegenüber Anwendungen anderer Isolationsumgebungen oder dem Wirtsystem nicht mehr ausgenutzt werden.

Schwachstelle mit SecurityManager geschlossen: Gleichbedeutend mit *Schwachstelle geschlossen*. Es muss jedoch ein SecurityManager in der Java-Laufzeitumgebung vorhanden sein, um die Schwachstelle zu schließen.

Schwachstelle mittels Überwachung vermeidbar: Dieser Status kann nur für Schwachstellen gelten, die einen Leistungseinbruch zur Folge haben. Durch Überwachung des Ressourcenverbrauchs einer Isolationsumgebung der jeweiligen Stufe kann ein Angriff erkannt werden. Durch die Zerstörung der Umgebung des Angreifers kann der Angriff gestoppt werden. Das Wirtsystem oder Anwendungen anderer Umgebungen werden durch die Zerstörung nicht beeinträchtigt. Da ein Angriff bereits begonnen haben muss, um ihn zu Erkennen, kann er durch die Überwachung prinzipiell nur abgebrochen jedoch nicht verhindert werden. Eine Ressourcenüberwachung ist im Isolationssystem bisher nicht implementiert, deshalb ist diese Gegenmaßnahme als hypothetisch anzusehen.

Es wurden die folgenden Isolationsstufen untersucht:

- Keine Isolation (siehe 5.1)
- Prozess (siehe 5.4)

Für die Stufe *Keine Isolation* wurde vorausgesetzt, dass der Benutzer die Anwendung über die Konsole des Haupt-Frameworks installiert, wie es auch ohne das Isolationssystem geschehen würde. Das Isolationssystem nimmt also keinerlei Einfluss auf die installierte Anwendung. Andere Vorbedingungen werden in der Beschreibungen der Schwachstellen erläutert.

Für die Evaluation wurde die Laufzeitumgebung des Sun Java Development Kit 6 Update 13 (Standard Edition) für Linux x64 verwendet. Als OSGi-Framework kam für beide Isolationsstufen die Referenzimplementierung Equinox [Eclc] in der Version 3.5 zum Einsatz. Als Betriebssystem wurde Debian GNU/Linux 5.0.3 (amd64) eingesetzt. Das evaluierte System wurde unter normalen Benutzerrechten ausgeführt (ohne root-Rechte). Für die Evaluation von Schwachstellen, die einen Leistungseinbruch zur Folge haben, ist die Leistungsfähigkeit der Hardware von Bedeutung: Es kam ein Computer mit dem Prozessor *Intel Core 2 Duo E4500* (2,2 GHz) und 2 GB Arbeitsspeicher zum Einsatz.

Die Beschreibungen der untersuchten Schwachstellen sind in Anhang A aufgeführt.

5.6.1 Fazit der Evaluierung der Effektivität von Isolationsstufen

In den Tabellen 5.2, 5.3 und 5.4 sind die Untersuchungsergebnisse nach den Angriffsarten zusammengefasst. Auf einige Schwachstellen treffen die Angriffsarten *Nichtverfügbarkeit* sowie *Leistungseinbruch* zu, weshalb sie in beiden zugehörigen Tabellen enthalten sind.

Zu jeder Schwachstelle wird der Status bezogen auf die jeweilige Isolationsstufe dargestellt. Dazu wird folgende Symbolik verwendet:

○ Schwachstelle offen

● Schwachstelle geschlossen

●^{S} Schwachstelle mit SecurityManager geschlossen

●^{Ü} Schwachstelle mittels Überwachung vermeidbar

Die Ergebnisse zur Angriffsart *Nichtverfügbarkeit* zeigen, dass ein Großteil der Schwachstellen durch die Isolationsstufe *Prozess* geschlossen werden können (siehe Tabelle 5.2). Im unisolierten Zustand sind die meisten Schwachstellen offen. Die drei Schwachstellen Runtime.exec.kill, System.exit und Runtime.halt können mit Hilfe der Stufe *Prozess* auch ohne die Verwendung eines Security Managers geschlossen werden. Das hat den Vorteil, dass auf den Security Manager verzichtet werden kann, wenn der Schutz vor diesen Schwachstellen im Vordergrund steht. Die Verwendung eines Security Managers bringt Leistungseinbuße mit sich, die in diesem Fall vermieden werden können.

Den meisten Schwachstellen der Angriffsart *Leistungseinbruch* kann nur durch Ressourcenüberwachung entgegengewirkt werden (siehe Tabelle 5.3). Dabei handelt es sich meist um Schwachstellen, deren Ausnutzung eine hohe CPU-Last zur Folge hat. Die Schwierigkeit bei der Ressourcenüberwachung besteht in diesen Fällen darin, zwischen einem Angriff und der

Name der Schwachstelle	Keine Isolation	Prozess
Management Utility Freezing - Infinite Loop	○	●^{Ü}
Management Utility Freezing - Thread Hanging	○	●
Excessive Size of Manifest File	●	●
Runtime.exec.kill	●^{S}	●
System.exit	●^{S}	●
Runtime.halt	●^{S}	●
Recursive Thread Creation	○	●^{Ü}
Hanging Thread	○	●
Infinite Loop in Method Call	○	●^{Ü}
Exponential Object Creation	●	●
Cycle Between Services	○	○
Blockieren der graphischen Benutzungsoberfläche	○	●

Tabelle 5.2: Schwachstellen der Angriffsart *Nichtverfügbarkeit*

normalen Nutzung der CPU durch die Anwendungen zu unterscheiden. Ein Ansatz ist die Anwendungen kurzzeitig zu stoppen und zu prüfen, ob die CPU-Last dadurch nachlässt. Damit lassen sich z. B. Angriffe erkennen, die eine Endlosschleife ausführen.

Die Hälfte der untersuchten Schwachstellen, die unberechtigte Zugriffe erlauben, können nur durch einen Security Manager geschlossen werden (siehe Tabelle 5.4). Wie bereits erwähnt, hat dies Leistungseinbuße zur Folge. Zudem kann in bestimmten Fällen ein Security Manager nur bedingt eingesetzt werden. Wenn eine Anwendung z. B. auf native Programmbibliotheken zurückgreifen muss, muss ihr die entsprechende Berechtigung eingeräumt werden, auch wenn dadurch zusätzliche Risiken entstehen, da die Java-Sicherheitsmechanismen bei nativem Code nicht mehr greifen.

Die drei Schwachstellen Code Observer, Component Data Modifier und Hidden Method Launcher, die den unberechtigten Zugriff auf Daten und Dienste anderer Anwendungen ermöglichen, können durch die Stufe *Prozess* vollständig geschlossen werden. Um die Schwach-

Name der Schwachstelle	Keine Isolation	Prozess
Management Utility Freezing - Infinite Loop	○	●Ü
Decompression Bomb	●	●
Sleeping Bundle	○	○
Big File Creator	○	●Ü
Memory Load Injection	○	●Ü
Stand Alone Infinite Loop	○	●Ü
Infinite Loop in Method Call	○	●Ü
Numerous Service Registration	○	●Ü

Tabelle 5.3: Schwachstellen der Angriffsart *Leistungseinbruch*

stellen auszunutzen, wird Reflection eingesetzt, die jedoch nicht über Prozessgrenzen hinweg funktioniert.

5.7 Evaluierung der Effizienz

Zur Evaluierung der Leistungseinbuße, die durch die Isolation entstehen, wurden zwei Aspekte untersucht. Einerseits wurden die Einbuße an Rechenleistung untersucht. Andererseits wurde der Aufwand zur Kommunikation zwischen isolierten Anwendungen und der Desktop-Grid-Plattform untersucht.

Dazu wurden Laufzeitmessungen mit den Isolationsstufen *Keine Isolation* und *Prozess* durchgeführt.

Zur Durchführung der Evaluierung kam ein Computer mit dem Prozessor *Intel Core 2 Duo E4500* (2,2 GHz) und 2 GB Arbeitsspeicher zum Einsatz. Es wurde die Laufzeitumgebung des Sun Java Development Kit 6 Update 13 (Standard Edition) für Linux x64 verwendet. Als OSGi-Framework kam für beide Isolationsstufen die Referenzimplementierung Equinox [Eclc] in der Version 3.5 zum Einsatz. Als Betriebssystem wurde Debian GNU/Linux 5.0.3 (amd64) eingesetzt.

Name der Schwachstelle	Keine Isolation	Prozess
Code Observer	●^{S}	●
Component Data Modifier	●^{S}	●
Hidden Method Launcher	○	●
Launch a hidden Bundle	●^{S}	●^{S}
Pirat Bundle Manager	○	●
Ausführung von schädlichem nativem Code	●^{S}	●^{S}
Unerlaubte Netzwerkzugriffe	○	●^{S}
Unerlaubte Zugriffe auf das Dateisystem	●^{S}	●^{S}

Tabelle 5.4: Schwachstellen der Angriffsart *Unberechtigter Zugriff*

5.7.1 Rechenleistung

Zur Untersuchung der Einbuße an Rechenleistung wurde eine Anwendung herangezogen, die hohe Anforderungen an die CPU-Leistung stellt. Dabei handelt es sich um eine Anwendung zur Darstellung der Mandelbrot-Menge [KWBT]. Es wird eine Abbildung mit 33600x33600 Bildpunkten erzeugt und die Dauer der Laufzeit gemessen. Die Anwendung verteilt die Berechnung auf mehrere Threads, so dass auch Prozessoren mit mehreren Kernen ausgelastet werden können.

Für jede Isolationsstufe wurden zehn Messungen durchgeführt. Die Mittelwerte der Ergebnisse sind in Abbildung 5.3 dargestellt. Der Mittelwert der Stufe *Kein Isolation* liegt bei 117,198 s, wobei das 95%-Konfidenzintervall [115,107; 119,289] beträgt. Für die Stufe *Prozess* liegt der Mittelwert bei 116,711 und das 95%-Konfidenzintervall beträgt [113,244; 120,177].

Zwischen dem nicht-isolierten Zustand und der Stufe *Prozess* ist kaum ein Unterschied zu erkennen. Das rührt daher, dass die Anwendung in beiden Fällen in einem eigenen Prozess ausgeführt wird. Die beiden Fälle unterscheiden sich lediglich in der Nähe zur Desktop-Grid-Plattform, was sich zwar auf den Kommunikationsaufwand auswirken kann, nicht jedoch auf die zur Verfügung stehende Rechenleistung. Da die untersuchte Anwendung nicht mit der Plattform kommuniziert, sind die Voraussetzung für beide Stufen gleich.

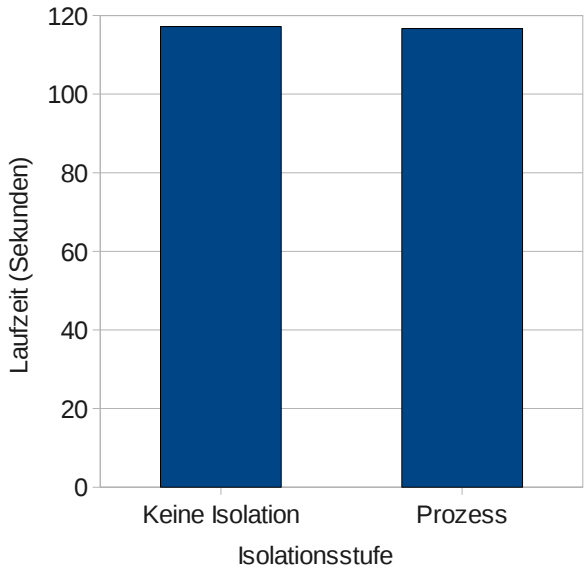

Abbildung 5.3: Laufzeitmessungen Rechenleistung

5.7.2 Kommunikationsaufwand

Zur Evaluierung des Kommunikationsaufwands wurden zwei Anwendungen implementiert: ein *Empfänger* und ein *Sender*. Der Empfänger fordert den Sender, über dessen Service, auf Daten zu senden. Daraufhin nutzt der Sender den Service des Empfängers, um eine große Anzahl an Zeichenketten an den Empfänger zu übermitteln. Der Empfänger gibt diese auf der Standardausgabe aus.

Dadurch wird z. B. die Nutzung eines zentralen Log-Service der Desktop-Grid-Plattform simuliert. Der Empfänger stellt den Log-Service dar und der Sender repräsentiert eine Anwendung die Log-Nachrichten ausgeben möchte.

Bei den Nachrichten handelt es sich um zufällig generierte Zeichenketten, mit einer durchschnittlichen Länge von 211 Zeichen (Min.: 18, Max.: 480) und einer Gesamtkapazität von 20 MB.

Die Laufzeitmessung wird vom Empfänger durchgeführt. Es wird die Dauer zwischen der Aufforderung zum Senden und dem Eintreffen der letzten Nachricht, deren Inhalt dem Empfänger bekannt ist, gemessen.

Für jede Isolationsstufe wurden 40 Messungen durchgeführt. Im Fall der Stufe *Keine Isolation* befinden sich Empfänger und Sender im Haupt-Framework. Für die Evaluierung der Stufe *Prozess* wurde der Empfänger im Haupt-Framework und der Sender in einer Isolationsumgebung der Stufe *Prozess* ausgeführt. Abbildung 5.4 zeigt die Mittelwerte der Messungen. Der Mittelwert der Stufe *Kein Isolation* liegt bei 2,772 s, wobei das 95%-Konfidenzintervall

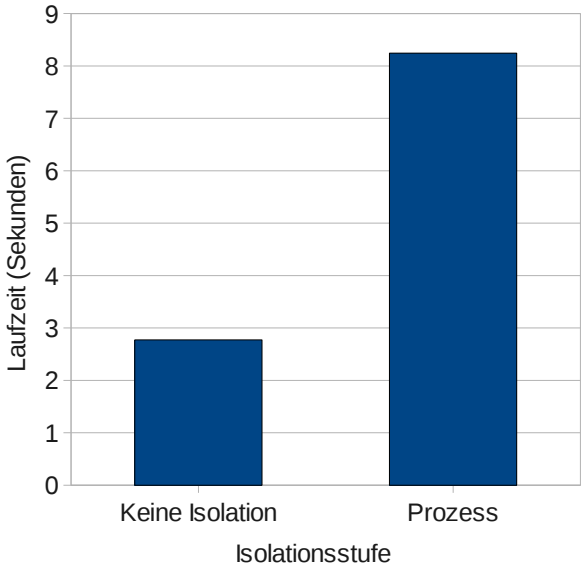

Abbildung 5.4: Laufzeitmessungen Kommunikationsaufwand

$[2,610; 2.934]$ beträgt. Für die Stufe *Prozess* liegt der Mittelwert bei $8,246$ und das 95%-Konfidenzintervall beträgt $[8,022; 8,470]$.

Im nicht-isolierten Zustand gleicht das Senden einem einfachen Methodenaufruf. Da lediglich Referenzen auf die gesendeten Nachrichten übergeben werden, entsteht nur wenig Aufwand und die Laufzeiten bleiben gering.

Bei den Ergebnissen der Isolationsstufe *Prozess* macht sich der zusätzliche Aufwand zur Übermittlung der Nachrichten über Remote Services bemerkbar. Zwischen Prozessen mit unterschiedlichen Adressräumen können keine Referenzen ausgetauscht werden, d. h. die Nachrichten müssen vom Speicher des Senders in den Speicher des Empfängers kopiert werden.

6 Isolationsvorgaben

Die Isolationsumgebung einer Desktop-Grid-Anwendung und deren Isolationsstufe sollen je nach Anwendungsfall angepasst werden können. Damit das möglich ist, müssen die Beteiligten Vorgaben zur Isolation der Anwendungen stellen können. Solche Vorgaben werden *Isolationsvorgaben* genannt.

Grundsätzlich kann zwischen zwei Arten von Isolationsvorgaben unterschieden werden:

- Vorgaben, die festlegen, welche Isolationsstufen für eine Anwendung genutzt werden dürfen
- Vorgaben, die sich auf die Nachbarschaft von Anwendungen beziehen

Ein *Nachbar* einer Anwendung ist eine andere Anwendung, die sich in derselben Isolationsumgebung befindet.

Isolationsvorgaben können von unterschiedlichen Beteiligten des Systems definiert werden. Beispielsweise könnte ein Nutznießer vorgeben, dass dessen Anwendung nicht mit einer Anwendung eines bestimmten anderen Nutznießers in der gleichen Isolationsumgebung ausgeführt wird. Ein Benutzer könnte für eine Anwendung eine bestimmte Isolationsstufe vorgeben, so dass er sein Wirtsystem vor Angriffen als geschützt erachtet.

Die Identifikation einer Anwendung geschieht über deren Eigenschaften, wie z. B. die Signaturen des zugehörigen Bundles. Eine Anwendung kann z. B. aufgrund einer fehlenden vertrauenswürdigen Signatur von der Nachbarschaft ausgeschlossen werden. Folgende Eigenschaften zur Verwendung in den Isolationsvorgaben sind denkbar:

- Das Bundle der Anwendung wurde nicht von einer bestimmten Organisation signiert.
- Das Bundle der Anwendung wurde von einer bestimmten Organisation signiert, die z. B. als nicht vertrauenswürdig erachtet wird.
- Das Bundle der Anwendung exportiert bestimmte Packages, z. B. Packages, die nur die Desktop-Grid-Plattform zur Verfügung stellen sollte.

Außerdem ist jede weitere Eigenschaft denkbar, die vor der Ausführung einer Anwendung bestimmt werden kann. Insbesondere können die, in der Manifest-Datei enthaltenen, Metadaten eines Bundles genutzt werden.

Aufgrund der Isolationsvorgaben müssen den Anwendungen Isolationsumgebungen zugeordnet werden. Dabei soll der Aufwand für die Isolation möglichst niedrig gehalten werden.

Je höher die Anzahl an Isolationsumgebungen und je höher der Isolationsgrad, desto höher der Aufwand für die Isolation. Das Ziel ist also, unter Berücksichtigung der Isolationsvorgaben, möglichst wenig Isolationsumgebungen mit möglichst geringem Isolationsgrad zu erzeugen. Somit handelt es sich um ein Optimierungsproblem. Die Lösung des Problems wird als das *Auflösen von Isolationsvorgaben* bezeichnet.

In Abschnitt 6.1 werden grundlegenden Typen von Isolationsvorgaben definiert. Im Abschnitt 6.2 wird beschrieben, wie erweiterte Isolationsvorgaben, aufgrund grundlegenden Vorgaben, definiert werden können.

Anschließend werden zwei Ansätze zur Auflösung von Isolationsvorgaben vorgestellt. Der erste Ansatz beschränkt sich bei den Vorgaben zu Nachbarschaftsbeziehungen auf den Ausschluss bestimmter Nachbarn (siehe 6.3). Zudem kann für eine Anwendung nur eine Isolationsstufe festgelegt werden. Im zweiten Ansatz werden auch zusätzliche Vorgaben berücksichtigt, die z. B. die Nachbarschaft bestimmter Anwendungen fordern (siehe 6.4).

6.1 Grundlegende Typen von Isolationsvorgaben

Es folgt eine Aufzählung von grundlegenden Isolationsvorgaben. In den folgenden Unterabschnitten wird diskutiert, ob diese Vorgaben sinnvoll eingesetzt werden können. Im Abschnitt 6.2 wird beschrieben, wie erweiterte Isolationsvorgaben, aufgrund der hier vorgestellten Vorgaben, definiert werden können.

- Anwendung a muss in einer anderen Isolationsumgebung ausgeführt werden als Anwendung b.
- Anwendung a muss in einer Isolationsumgebung einer anderen Isolationsstufe ausgeführt werden als Anwendung b.
- Eine Anwendung muss in einer Isolationsumgebung ausgeführt werden, die einer Isolationsstufe aus einer bestimmten Menge von Isolationsstufen angehört.
- Anwendung a muss mit Anwendung b in einer Isolationsumgebung ausgeführt werden.

6.1.1 Unterschiedliche Isolationsumgebungen

Die Ausführung zweier Anwendungen in unterschiedlichen Isolationsumgebungen stellt die schwächste Vorgabe zur Isolation von Anwendungen dar. Ohne zusätzliche Vorgaben zur Isolationsstufe oder Kenntnisse über die verfügbaren Isolationsumgebungen des Systems ist sie nur bedingt sinnvoll. Allein einsetzbar ist sie z.B., wenn eine schwache Isolation gefordert wird. Zusätzlich sollen dem System aber möglichst große Freiheitsgrade betreffend der Zuweisung von Isolationsstufen gewährt werden, um mit einer möglichst niedrigen Anzahl an Isolationsumgebungen aus zu kommen.

6.1.2 Unterschiedliche Isolationsstufen

Mit der Vorgabe zur Ausführung zweier Anwendungen in unterschiedlichen Isolationsstufen, verhält es sich ähnlich, wie mit der vorigen Vorgabe. Allein eingesetzt führt sie zwangsläufig dazu, dass die Anwendungen in unterschiedlichen Umgebungen ausgeführt werden. Da die Umgebungen unterschiedlichen Isolationsstufen angehören, wird eine der Anwendungen stärker isoliert als die Andere. Mehr Aussagen lassen sich im Allgemeinen nicht treffen. Es ist zum Beispiel nicht vorherzusehen, welche der beiden Anwendungen in der stärkeren Isolationsstufe ausgeführt wird. Diese Isolationsvorgabe lässt sich auch nicht sinnvoll mit anderen Vorgaben kombinieren. Deshalb wurde sie bei der Umsetzung des Systems nicht in Betracht gezogen.

6.1.3 Vorgabe von Isolationsstufen

Die Vorgabe von bestimmten Isolationsstufen für eine Anwendung lässt sich wiederum sinnvoll nutzen. Denn damit wird der Isolationsgrad einer Anwendung festgelegt. Da eine Anwendung in nur genau einer Umgebung ausgeführt werden kann, welche wiederum genau einer Isolationsstufe angehört, reicht die Vorgabe genau einer Stufe eigentlich aus. Jedoch herrschen in Desktop Grids oft heterogenen Bedingungen, wodurch nicht immer auf allen Wirtsystemen die gleiche Stufen zur Verfügung stehen. In einem solchen Fall kann eine Menge von Stufen vorgegeben werden, die für eine Anwendung in Frage kommen. Auf den Wirtsystemen wird dann eine verfügbare Stufe aus dieser Menge genutzt. Ein weiterer Anwendungsfall für die Vorgabe mehrerer Stufen, ist die Vorgabe eines Mindestisolationsgrads für eine Anwendung. Der Betreiber eines Desktop Grids möchte z. B., dass alle Anwendungen einen bestimmten Isolationsgrad nicht unterschreiten. Wird nur die Isolationsstufe angegeben, die diesem Isolationsgrad entspricht, werden höhere Isolationsstufen ausgeschlossen. Einem Nutznießer wäre dann die Ausführung seiner Anwendung in einer stärkeren Isolationsstufe nicht mehr möglich. Erlaubt der Betreiber jedoch auch die Ausführung in den höheren Isolationsstufen, entsteht dieses Problem nicht.

6.1.4 Gemeinsame Isolationsumgebung

Mit der Vorgabe zur Ausführung von Anwendungen in derselben Isolationsumgebung wird der Isolation zwar entgegengewirkt, jedoch ist diese Vorgabe dennoch sinnvoll. Liegt der Anwendungsfall vor, dass zwei Anwendungen häufig miteinander kommunizieren, ist es von Vorteil diese Anwendungen in derselben Isolationsumgebung auszuführen. Dadurch wird der Kommunikationsaufwand gering gehalten, da keine zusätzlichen Mechanismen zur entfernten Kommunikation benötigt werden. Die Anwendungen können über OSGi-

Services direkt miteinander kommunizieren, da sie in einem gemeinsamen Kind-Framework ausgeführt werden. Die Nutzung eines Service entspricht dann einem einfachen Java-Methodenaufruf.

6.2 Erweiterte Isolationsvorgaben

Auf Basis der grundlegenden Isolationsvorgaben lassen sich weitere Isolationsvorgaben definieren. Dazu werden die grundlegenden Vorgaben miteinander kombiniert oder bei der Auflösung der Isolationsvorgaben mit Informationen angereichert, die erst zur Laufzeit zur Verfügung stehen, z. B. die auf dem Wirtsystem verfügbaren Isolationsstufen.

Der Benutzer oder ein anderer Beteiligter definiert die erweiterten Isolationsvorgaben, welche vor der Auflösung der Isolationsvorgaben in grundlegenden Vorgaben übersetzt werden. Dadurch müssen bei der Lösung des Optimierungsproblems nur die grundlegenden Vorgaben berücksichtigt werden und es wird unnötige Komplexität vermieden.

6.2.1 Isolation der Adressräume/des Massenspeichers

Zwei Anwendungen sollen nicht im selben Adressraum ausgeführt werden. Das hat zur Folge, dass die Anwendungen in unterschiedlichen Prozessen ausgeführt werden müssen. Erreicht werden kann das, indem die Vorgaben gestellt werden, dass die Anwendungen in der Isolationsstufe *Prozess* ausgeführt werden und die Anwendungen sich in unterschiedlichen Isolationsumgebungen befinden müssen. Dadurch wird jede Anwendung in einem eigenen Prozess ausgeführt, die ihre eigenen Adressräume besitzen.

Zur Isolation des Massenspeichers, also z. B. der Festplatte, auf den die Anwendungen Zugriff haben, kann ähnlich wie bei der Isolation der Adressräume vorgegangen werden. Nur wird statt der Isolationsstufe *Prozess*, eine Stufe zur Systemvirtualisierung genutzt, die jeder Anwendung ein eigenes virtuelles Laufwerk zur Verfügung stellt.

6.2.2 Vorgabe von Systemvoraussetzungen

Eine Anwendung soll in einem bestimmten Betriebssystem, einer bestimmten Java-Laufzeitumgebung oder einer bestimmten OSGi-Implementierung ausgeführt werden. Es sollen also bestimmte *Systemvoraussetzungen* einer Anwendung erfüllt werden.

Es gibt zwei Ansätze diese Vorgabe zu definieren. Eine Möglichkeit ist es, für die benötigten Betriebssysteme, Laufzeitumgebungen und OSGi-Frameworks eigene Isolationsstufen zu definieren. Läuft auf dem Wirtsystem z. B. Linux und es soll eine Anwendung in Mac OS ausgeführt werden, wird die Vorgabe gestellt, dass sie in der Stufe *Virtuelle Maschine Mac OS* ausgeführt werden soll. Für jedes andere Betriebssystem existieren weitere Isolationsstufen.

Algorithmus 6.1 Vorgabe von Systemvoraussetzungen

A: Menge der Anwendungen
I: Menge der Isolationsvorgaben (besteht nur aus grundlegenden Vorgaben)
S: Menge der Isolationsstufen
U: Menge der Isolationsumgebungen
$f : A \to U$ // Abbildung von Anwendungen auf Isolationsumgebungen (Ergebnis der Auflösung der Vorgaben)
// Übersetzen der Vorgaben von Systemvoraussetzungen in grundlegende Isolationsvorgaben
for all $a \in A$ **do**
  if a hat bestimmte Systemvoraussetzungen **then**
    // Anwendung a auf geeignete Isolationsstufen beschränken
    $S' \leftarrow \{s \in S \mid s$ kann Systemvoraussetzungen erfüllen$\}$
    $I \leftarrow I \cup \{(a$ wird in einer Stufe aus der Menge S' ausgeführt$)\}$
    // Konflikte mit Systemvoraussetzungen anderer Anwendungen vorbeugen
    $A' \leftarrow \{b \in A \mid b$ hat Systemvoraussetzungen, die sich mit a widersprechen$\}$
    $I \leftarrow I \cup \{(a$ ist in einer anderen Umgebungen als $b) \mid b \in A'\}$
  end if
end for
// Auflösen der Isolationsvorgaben
$f \leftarrow$ löseIsolationsvorgaben(A, I, S, U)
$g \leftarrow f^{-1}$ // $g : U \to \mathcal{P}(A)$ ist Umkehrfunktion von f
// Erzeugen der Isolationsumgebungen
for all $u \in U$ **do**
  if $|g(u)| > 0$ **then** // Wenn u Anwendungen enthält...
    Erzeuge Umgebung, die möglichst allen Systemvoraussetzungen der Anwendungen $g(u)$ entsprechen
    Führe die Anwendungen aus der Menge $g(u)$ in u aus, deren Systemvoraussetzungen entsprochen werden kann
  end if
end for

Zieht man noch unterschiedliche Versionen und die Kombination verschiedener Laufzeitumgebungen mit unterschiedlichen Betriebssystemen in Betracht, wird jedoch schnell eine hohe Anzahl an Stufen erreicht. Das kann sich negativ auf die Laufzeit bei der Auflösung von Isolationsvorgaben auswirken. Außerdem weisen Isolationsstufen, die sich nur im Betriebssystem oder der Java-Laufzeitumgebung unterscheiden, in manchen Fällen kaum unterschiedliche Isolationseigenschaften auf. Das widerspricht aber dem Begriff der Isolationsstufe, der die Möglichkeit zur totale Ordnung der Stufen impliziert. Deshalb ist dieser Ansatz nicht besonders geeignet.

Ein weitere Möglichkeit ist es, bei der Auflösung der Isolationsvorgaben die Tatsache zu ignorieren, dass es unterschiedliche Betriebssysteme, Laufzeitumgebungen und Frameworks gibt. Erst beim Erzeugen der Isolationsumgebungen, wird überprüft, welche Systemvoraussetzungen die zugewiesenen Anwendungen fordern und die Isolationsumgebung entsprechend angepasst. Wird z. B. ein bestimmtes Betriebssystem vorausgesetzt, muss die Umgebung natürlich einer Stufe angehören, die Systemvirtualisierung unterstützt. Also müssen Anwendungen mit speziellen Systemvoraussetzungen auf Isolationsstufen beschränkt werden, die an die Voraussetzungen angepasst werden können. Das ist mit Hilfe der grundlegenden Vorgaben möglich. Jedoch reicht das im Allgemeinen nicht aus, da zwei Anwendungen, die widersprüchliche Voraussetzungen haben, also z. B. unterschiedliche Betriebssysteme benötigen, derselben Isolationsumgebung zugewiesen werden können. Es können aber in einer Umgebung nicht unterschiedliche Betriebssysteme zur Verfügung gestellt werden. Einem solchen Konflikt kann jedoch vorgebeugt werden, indem für alle Anwendungen, deren Systemvoraussetzungen sich widersprechen, zusätzlich die Vorgabe gestellt wird, dass sie sich in unterschiedlichen Umgebungen befinden müssen. Auch das kann mit Hilfe der grundlegenden Vorgaben bewerkstelligt werden.

Algorithmus 6.1 fasst die oben beschriebene Vorgehensweise in Pseudocode zusammen.

6.3 Auflösung der Isolationsvorgaben auf Basis des Knotenfärbungsproblems

Dieser Abschnitt führt das Problem zur Auflösung von Isolationsvorgaben auf das Knotenfärbungsproblem zurück. Dieser Ansatz ist beschränkt auf zwei Arten der grundlegenden Isolationsvorgaben. Es können Vorgaben gestellt werden, die die Nachbarschaft zwischen Anwendungen ausschliessen. Zudem muss für jede Anwendung genau eine Isolationsstufe vorgegeben werden.

Da durch die Vorgaben bereits für jede Anwendung eine Isolationsstufe bekannt ist, beschränkt sich die Auflösung der Vorgaben darauf, die Anwendungen jeder Stufe den Isolationsumgebungen zuzuweisen. Die Anzahl der Isolationsumgebungen soll dabei möglichst gering gehalten werden. Es handelt sich also um ein Optimierungsproblem, bei dem die zu minimierende Bewertungsfunktion die Anzahl der Isolationsumgebungen ist.

Dieses Problem lässt sich auf die Knotenfärbung der Graphentheorie zurückführen. Eine Knotenfärbung weist den Knoten eines Graphen Farben zu, so dass benachbarte Knoten unterschiedliche Farben besitzen.

Definition 6.3.1 (Knotenfärbung [Die06]) *Gegeben ist ein ungerichteter Graph $G = (V, E)$ mit der Knotenmenge V und der Kantenmenge E. Des Weiteren ist eine (endliche) Menge von Farben F gegeben. Gesucht ist eine Abbildung $f : V \to F$, so dass für alle Knotenpaare $\{u, v\} \in E$ gilt $f(u) \neq f(v)$.*

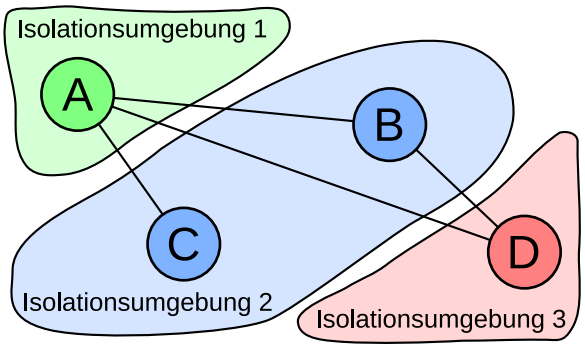

Abbildung 6.1: Zusammenhang zwischen Knotenfärbung und Auflösung von Isolationsvorgaben

Fasst man die Knotenfärbung als ein Optimierungsproblem auf, so ist eine Abbildung f gesucht, die mit möglichst wenig Farben auskommt. Die zu minimierende Bewertungsfunktion ist also die Anzahl der verwendeten Farben aus der Menge F.

6.3.1 Zurückführung auf das Knotenfärbungsproblem

Die Auflösung von Isolationsvorgaben lässt sich folgendermaßen auf das Knotenfärbungsproblem zurückführen. Für jeden Isolationsstufe wird ein Graph erstellt, dessen Knotenfärbungsproblem zu lösen ist. Die beteiligten Mengen aus Definition 6.3.1 werden folgendermaßen abgebildet.

- Die Anwendungen einer Isolationsstufe bilden die Knotenmenge eines Graphen.
- Der Auschluss der Nachbarschaft zwischen einer Anwendung a_1 und einer Anwendung a_2 wird durch eine Kante $\{a_1, a_2\}$ zwischen den jeweiligen Knoten dargestellt.
- Eine Farbe $c \in F$ entspricht einer Isolationsumgebung der jeweiligen Isolationsstufe.

Das Ergebnis der Färbung wird folgendermaßen interpretiert: Alle Anwendungen deren zugehörige Knoten die selbe Farbe haben, werden in der selben Isolationsumgebung ausgeführt.

Damit lassen sich existierende Knotenfärbungsalgorithmen für die Auflösung von Isolationsvorgaben einsetzen.

Abbildung 6.1 zeigt ein Beispiel für die Auflösung folgender Isolationsvorgaben:

- Anwendung A möchte nicht Nachbar von B, C sein
- Anwendung B möchte nicht Nachbar von A sein
- Anwendung D möchte nicht Nachbar von A, B sein

6.3.2 NP-Vollständigkeit

Das Problem der Auflösung von Isolationsvorgaben bei einer vorgegebene Anzahl von Isolationsumgebungen ist NP-vollständig.

Definition 6.3.2 (NP-Vollständigkeit [Sch01]) *Eine Sprache S heißt* NP-hart, *falls für alle Sprachen $L \in NP$ gilt: L ist polynomial reduzierbar auf S.*

Eine Sprache S heißt NP-vollständig, *falls S NP-hart ist und $S \in NP$ gilt.*

Dass das Problem in NP liegt, lässt sich durch Konstruktion einer nichtdeterministischen Turingmaschine zeigen: Zunächst „rät“ die nichtdeterministische Maschine eine Abbildung von Anwendungen zu Isolationsumgebungen. Anschließend muss festgestellt werden, ob die Abbildung den Isolationsvorgaben entspricht. Ist dies der Fall, geht die Maschine in einen akzeptierenden Endzustand über.

Diese Überprüfung kann folgendermaßen umgesetzt werden. A ist die Menge der Anwendungen. I ist die Menge der Isolationsvorgaben. $\{a_1, a_2\} \in I$ entspricht dem Ausschluss der Nachbarschaft zwischen den Anwendungen a_1 und a_2. Die Menge U enthält die Isolationsumgebungen. $g : A \rightarrow U$ ist die Abbildung von Anwendungen auf Isolationsumgebungen.

```
procedure VerifyIsolation(A,I,U,g)
    for all {a,b} ∈ I do
        if g(a) = g(b) then
            return FALSE
        end if
    end for
    return TRUE
end procedure
```

Entspricht die gegebene Abbildung g den Isolationsvorgaben, wird $TRUE$ zurückgegeben, ansonsten $FALSE$. Da die For-Schleife k-mal durchlaufen wird, mit $k = |I|$ und $|I| \in \mathbb{N}$, ist die Rechenzeit des Algorithmus polynomial. Also liegt das Problem der Auflösung von Isolationsvorgaben in NP.

Dass das Problem NP-hart ist, lässt sich durch die Umkehrung der Reduzierung aus Abschnitt 6.3.1 zeigen. Denn das Entscheidungsproblem, ob eine Knotenfärbung mit einer bestimmten Anzahl an Farben für einen Graph existiert, ist NP-vollständig. Das Knotenfärbungsproblem kann in polynomialer Rechenzeit auf das Problem der Auflösung von Isolationsvorgaben reduziert werden.

Die Mengen und die Abbildung g werden, wie folgt auf das Knotenfärbungsproblem abgebildet:

- Die Knoten V des Graphen entsprechen den Anwendungen.

- Eine Kante $\{u, v\} \in E$ wird durch den Auschluss der Nachbarschaft zwischen einer Anwendung u und einer Anwendung v dargestellt.
- Eine Isolationsumgebung entspricht einer Farbe.

Diese Abbildung ist in polynomialer Rechenzeit durchführbar. Also ist das Problem der Auflösung von Isolationsvorgaben NP-hart. Da schon gezeigt wurde, dass das Problem in NP liegt, kann geschlossen werden, dass es NP-vollständig ist.

6.4 Formulierung der Auflösung von Isolationsvorgaben als Constraint Optimization Problem

Im Gegensatz zum vorangegangen Ansatz (siehe 6.3) werden in diesem Ansatz auch Isolationsvorgaben berücksichtigt, die über den Auschluss von Nachbarschaftsbeziehungen hinausgehen. Zudem muss einer Anwendung nicht explizit eine bestimmte Isolationsstufe zugewiesen werden, was die Berücksichtigung von Vorgaben aus unterschiedlichen Quellen erlaubt. Allerdings ist dadurch auch nicht mehr von vornherein die Isolationsstufe einer Anwendung bekannt, weshalb ein Algorithmus zur Knotenfärbung auf das vorliegende Problem nicht ohne Weiteres anwendbar ist. Jedoch lässt sich die Zuweisung von Isolationsumgebungen und -stufen als Constraint Optimization Problem formulieren. Die folgende Definition ist stark komprimiert. Eine ausführliche Definition kann dem angegebenen Buch entnommen werden.

Definition 6.4.1 (Constraint Optimization Problem [Tsa93]) *Ein Constraint Optimization Problem (COP) besteht im Wesentlichen aus zwei Teilen:*

- *Constraint Satisfaction Problem*
  - *Viewpoint $V = (Z, D)$ [LLo6]*
    - *endliche Menge von Variablen Z, z. B. $\{x, y\}$*
    - *Abbildung D der Variablen auf deren Wertebereiche, wobei D_x den Wertebereich der Variable x bezeichnet, z. B. $D_x = \{1, 2, 3\}, D_y = \{1, 2, 3\}$*
  - *endliche Menge an Constraints (Bedingungen) C, z. B. $\{(x < y)\}$*
  - *Eine Lösung eines Constraint Satisfaction Problems ist ein Belegung der Variablen unter Berücksichtigung der Wertebereiche und Constraints, z. B. $(x = 1, y = 2)$.*
- *Bewertungsfunktion f, die jeder Lösung auf Basis der Variablenbelegung einen numerischen Wert zuordnet, z. B. $f(x, y) = 2x - y$*

Das Ziel ist eine Lösung mit optimalem Wert der Bewertungsfunktion zu finden. Der optimale Wert ist, je nach Anwendungsfall, der minimale oder maximale Wert der Bewertungsfunktion. Für das obige Beispiel ist $(x = 1, y = 3)$ die Lösung mit dem minimalen Wert -1. Die Lösung, mit dem maximalen Wert 1 der Bewertungsfunktion, ist $(x = 2, y = 3)$.

Der grundlegende Wertebereich ist nicht definiert. Für die Problemformulierung können also Variablen beliebigen Typs verwendet werden, z. B. Variablen der reellen oder natürlichen Zahlen oder Mengenvariablen. Jedoch werden von vielen Programmbibliotheken zur Lösung von Constraint Optimization Problems die natürlichen Zahlen am Besten unterstützt [cho, Tam, Hebo8, TTBo8]. Deshalb sind die Wertebereiche im Folgenden auch aufgrund der natürlichen Zahlen definiert.

Im Folgenden werden zwei Möglichkeiten beschrieben die Zuordnung von Isolationsumgebungen als Constraint Optimization Problem zu formulieren. Beiden Fällen liegt das gleiche Prinzip zugrunde. Die Isolationsvorgaben werden als Constraints formuliert. Die zu minimierende Bewertungsfunktion ist die Anzahl an Isolationsumgebungen.

Die Unterschiede lassen sich bereits an den Viewpoints der beiden Formulierungen erkennen, also an der Nutzung der Variablen und deren Interpretation. Die erste Formulierung definiert eine kleinere Anzahl an Variablen und Constraints im Vergleich zur anderen Formulierung (siehe 6.4.1). Jedoch wird u.a. das *nvalue* Constraint verwendet. Mit Hilfe dessen kann die Anzahl an unterschiedlichen Werten in den Variablen eingeschränkt werden. Jedoch wird dieses Constraint nicht von allen Programmbibliotheken zur Verfügung gestellt, z. B. nicht von Cream 1.0.6 [Tam]. Bei dieser Formulierung lassen sich, durch eine Anpassung des Suchraums, auch für höhere Anzahlen an Anwendungen akzeptable Laufzeiten erzielen.

Die zweite Formulierung kommt mit Constraints basierend auf Vergleichsrelationen ($=$, $>$, $\neq$) aus (siehe 6.4.2). Jedoch ist die Laufzeit nur bei einer niedrigen Anzahl an Anwendungen akzeptabel. Die Variablen sind als Matrix mit zwei Dimensionen angeordnet (x_{ij}). Deshalb wird diese Formulierung im Folgenden als *zweidimensionaler Viewpoint* bezeichnet. Die andere Formulierung wird im Gegenzug *eindimensionaler Viewpoint* genannt, denn dort sind die Variablen als Vektor angeordnet (x_i).

Im Folgenden werden die beiden Viewpoints detailliert beschrieben. Die Bewertungsfunktion wird in einem eigenen Abschnitt erklärt, da sie für beide Viewpoints anwendbar ist (siehe 6.4.3). Anschließend werden zwei Verfahren zur Einschränkung der Wertebereiche vorgestellt (siehe 6.4.4). Das Ziel der Verfahren ist der Ausschluss von Werten, von denen bereits vor der Lösungssuche bekannt ist, dass sie zu ungültige Belegungen oder nicht optimalen Lösungen führen. Dadurch wird der Suchraum verkleinert, was die Lösungssuche beschleunigen kann. In Abschnitt 6.5.1.3 werden Verfahren vorgestellt, die das Auftreten von Symmetrien vermeiden sollen. Durch Symmetrien entstehen viele äquivalente Lösungen, die den Suchraum unnötig vergrößern. Schlussendlich werden die Viewpoints hinsichtlich der Laufzeit bei der Lösungssuche evaluiert (siehe 6.5).

6.4.1 Eindimensionaler Viewpoint

6.4.1.1 Variablen

Für jede Anwendung i existiert eine Variable x_i, deren Wert die Identifikationsnummer (ID) der Isolationsumgebung ist, in der die Anwendung enthalten ist. Für i gilt: $1 \leq i \leq m$, wobei m die Anzahl an Anwendungen darstellt.

6.4.1.2 Wertebereich

Der Wertebereich einer Variable x_i ist unterteilt in n (nicht-überlappende) Teilintervalle. Jedes Teilintervall entspricht den IDs einer Isolationsstufe. n ist die Anzahl an Isolationsstufen. Es kann höchstens so viele Isolationsumgebungen wie Anwendungen geben, d. h. die Länge eines Teilintervalls ist durch die Anzahl m an Anwendungen nach oben hin begrenzt. Für die Isolationsstufe *Keine Isolation* gilt die Besonderheit, dass sie nur genau eine Isolationsumgebung besitzt.

Der Wertebereich von x_i ist also definiert durch:

$$D_{x_i} = [1; l] \subset \mathbb{N} \mid l = 1 + (n-1) \cdot m$$

l entspricht der Anzahl an IDs für Isolationsumgebungen.

Die Wertebereich beginnt bei $x_i = 1$ mit der Stufe $j = 1$: *Keine Isolation*. Danach folgen die Teilintervalle der anderen Isolationsstufen $j > 1$ aufsteigend geordnet nach dem Isolationsgrad. Das heißt, auf die Stufe $j = 1$ folgt die Stufe $j = 2$ mit der schwächsten Isolation. Das letzte Teilintervall $j = n$ ist der Stufe mit der stärksten Isolation zugeordnet.

Ein Intervall ID_j mit IDs der Umgebungen der Stufe j ist also folgendermaßen definiert, wobei gilt $1 \leq j \leq n$:

$$\mathrm{ID}_j = \begin{cases} [1;1] \subset \mathbb{N}, & \text{wenn } j = 1 \text{ (Keine Isolation)} \\ [\sup(\mathrm{ID}_{j-1} + 1);\ \sup(\mathrm{ID}_{j-1}) + m] \subset \mathbb{N}, & \text{sonst} \end{cases}$$

Die Rekursion kann wie folgt aufgelöst werden.

$$\mathrm{ID}_j = \begin{cases} [1;1] \subset \mathbb{N}, & \text{wenn } j = 1 \text{ (Keine Isolation)} \\ [(j-2) \cdot m + 2;\ (j-1) \cdot m + 1] \subset \mathbb{N}, & \text{sonst} \end{cases}$$

Tabelle 6.1 zeigt eine gültige Belegung der Variablen x_i unter der Annahme das drei Isolationsstufen existieren: 1. Keine Isolation, 2. Prozess, 3. Virtuelle Maschine (VM). Da es sich um vier Anwendungen handelt, lauten die zugehörigen Teilintervalle von D_{x_i} wie folgt: 1. Stufe: $[1;1]$, 2. Stufe: $[2;5]$, 3. Stufe: $[6;9]$. Demnach befinden sich die Anwendungen A

Anwendung	A	B	C	D
Variable	x_1	x_2	x_3	x_4
Wert	1	1	6	7

Tabelle 6.1: Beispielbelegung der Variablen x_i

und B in der Isolationsumgebung der Stufe *Keine Isolation*. Den Anwendungen C und D sind unterschiedliche Isolationsumgebungen der Stufe *VM* zugeordnet.

6.4.1.3 Isolationsvorgaben

Isolationsvorgaben werden als Constraints formuliert. Im Folgenden sind die Constraints für die grundlegenden Isolationsvorgaben dargestellt.

Die Constraints eines Constraint Optimization Problems müssen alle erfüllt sein, damit eine Lösung gültig ist. Das heißt, die Constraints bilden eine logische Konjunktion (UND-Verknüpfung). *Reified Constraints* erlauben auch die Verwendung von logischen Diskunktionen (ODER-Verknüpfung).

> **Definition 6.4.2 (Reified Constraint [RBW06])** *Ein Reified Constraint verknüpft ein Constraint mit einer binären Variable. Ist das Constraint erfüllt, nimmt die Variable den Wert* 1 *an. Ist das Constraint nicht erfüllt, hat die Variable den Wert* 0*. Dadurch lassen sich logische Disjunktion bilden. Seien z.B.* C_1 *und* C_2 *Constraints, die mittels Reified Constraints mit den binären Variablen* c_1 *und* c_2 *verbunden sind. Die logische Disjunktion* $(C_1 \vee C_2)$ *lässt sich mit Hilfe der Variablen durch folgendes Constraint ausdrücken:* $(c_1 + c_2 > 0)$.

Da nicht jede Programmbibliothek Reified Constraints anbietet, werden auch Alternativen angegeben, die logische Ausdrücke mit Hilfe der Betragsfunktion und der Signum-Funktion definieren.

Anwendung a ist in einer anderen Isolationsumgebung als Anwendung b:

$$x_a \neq x_b$$

Anwendung a ist in einer Isolationsumgebung der Isolationsstufe k oder l:

$$\big(x_a \geq \inf(\mathrm{ID}_k) \wedge x_a \leq \sup(\mathrm{ID}_k)\big) \vee \big(x_a \geq \inf(\mathrm{ID}_l) \wedge x_a \leq \sup(\mathrm{ID}_l)\big)$$

Die Alternative ohne Nutzung von logischen Verknüpfungen lautet:

$$
\begin{aligned}
&\big(x_a > \inf(\mathrm{ID}_k) - 1 \wedge \sup(\mathrm{ID}_k) + 1 > x_a\big) \\
&\vee\big(x_a > \inf(\mathrm{ID}_l) - 1 \wedge \sup(\mathrm{ID}_l) + 1 > x_a\big) \\
\rightsquigarrow\quad &\big(\mathrm{sgn}(x_a - (\inf(\mathrm{ID}_k) - 1)) = 1 \wedge \mathrm{sgn}((\sup(\mathrm{ID}_k) + 1) - x_a) = 1\big) \\
&\vee\big(\mathrm{sgn}(x_a - (\inf(\mathrm{ID}_l) - 1)) = 1 \wedge \mathrm{sgn}((\sup(\mathrm{ID}_l) + 1) - x_a) = 1\big) \\
\rightsquigarrow\quad &\big(\mathrm{sgn}(x_a - \inf(\mathrm{ID}_k) + 1) + \mathrm{sgn}(\sup(\mathrm{ID}_k) + 1 - x_a) = 2\big) \\
&\vee\big(\mathrm{sgn}(x_a - \inf(\mathrm{ID}_l) + 1) + \mathrm{sgn}(\sup(\mathrm{ID}_l) + 1 - x_a) = 2\big) \\
\rightsquigarrow\quad &\Big|\mathrm{sgn}\Big(\big|\mathrm{sgn}(x_a - \inf(\mathrm{ID}_k) + 1) + \mathrm{sgn}(\sup(\mathrm{ID}_k) + 1 - x_a) - 2\big|\Big) - 1\Big| \\
&+\Big|\mathrm{sgn}\Big(\big|\mathrm{sgn}(x_a - \inf(\mathrm{ID}_l) + 1) + \mathrm{sgn}(\sup(\mathrm{ID}_l) + 1 - x_a) - 2\big|\Big) - 1\Big| > 0
\end{aligned}
$$

Anwendung a ist in derselben Isolationsumgebung wie Anwendung b:

$$x_a = x_b$$

6.4.1.4 Anzahl an Isolationsumgebungen

Zur Bewertung einer Lösung muss u.a. die Anzahl an Isolationsumgebungen z bestimmt werden. Dazu kann das *nvalue* Constraint verwendet werden.

Definition 6.4.3 (Number of Distinct Values Constraint [Bel01]) *Das Constraint $\mathrm{nvalue}(v_0, \{v_1, ..., v_n\})$ ist erfüllt, wenn v_0 die Anzahl an eindeutigen Werten enthält, die die Variablen $v_1, ..., v_n$ angenommen haben.*

Haben die Variablen $(v_1, ..., v_5)$ z. B. die Werte $(4, 2, 2, 2, 4)$ angenommen, muss $v_0 = 2$ gelten, damit das Constraint erfüllt ist. Denn es kommen genau zwei eindeutigen Werte (2 und 4) in den Variablen vor.

Angewendet auf die Variablen x_i ergibt es die Anzahl an Isolationsumgebungen. Sei m die Anzahl an Anwendungen und z eine Variable mit dem Wertebereich $[1, m] \subset \mathbb{N}$. Dann ist durch das folgende Constraint festgelegt, dass z_{ges} die Anzahl an Umgebungs-IDs, denen die Anwendungen 1 bis m zugeordnet sind, enthält. Dies entspricht genau der Gesamtanzahl an Isolationsumgebungen.

$$\mathrm{nvalue}(z_{ges}, \{x_1, ..., x_m\})$$

Dieses Constraint berücksichtigt jedoch nicht die Tatsache, dass die Isolationsumgebung der ersten Isolationsstufe keinen zusätzlichen Ressourcenverbrauch verursacht. Die einzige Umgebung der ersten Stufe *Keine Isolation*, das Haupt-Framework, existiert auch, wenn keine

Anwendung darin ausgeführt wird. Es verursacht also keinen zusätzlichen Ressourcenverbrauch, eine Anwendung in dieser Stufe auszuführen. Deshalb sollte die Umgebung der Stufe auch nicht in der Anzahl der Isolationsumgebungen berücksichtigt werden. Dem wird durch folgende Constraints Rechnung getragen:

$$\text{nvalue}(z', \{x_1, ..., x_m, 1\})$$
$$z = z' - 1$$

Dadurch wird erreicht, dass die Umgebung der ersten Stufe, die die Umgebungs-ID 1 hat, in z' immer mitgezählt wird, auch wenn sie von keiner Anwendung genutzt wird. Anschließend wird sie wieder abgezogen, so dass z die Anzahl an Isolationsumgebungen, ohne die Umgebung der Stufe *Keine Isolation*, enthält.

Alternativ kann das *Global Cardinality Constraint* (GCC) zur Zählung der Isolationsumgebungen verwendet werden.

Definition 6.4.4 (Global Cardinality Constraint [BCR05]) *Das Constraint hat die Form* $gcc(\{v_1, ..., v_m\}, \{u_1, ..., u_n\})$. $\{v_1, ..., v_m\}$ *ist eine Menge von Variablen.* $\{u_1, ..., u_n\}$ *ist eine Menge von Variablentupeln. Es gilt* $u_i = (w_i, a_i)$. *Das Constraint ist erfüllt, wenn jede Variable* a_i *die Anzahl der Vorkommnisse des Werts* w_i *in den Variablen* $v_1, ..., v_n$ *enthält.*

Haben die Variablen $(v_1, ..., v_5)$ *die Werte* $(4, 2, 2, 2, 4)$ *angenommen und es gilt* $w_1 = 2$ *und* $w_2 = 4$, *dann muss* $a_1 = 3$ *und* $a_2 = 2$ *gelten, damit das Constraint erfüllt ist.*

Sei m die Anzahl an Anwendungen. l sei die größte ID für Isolationsumgebungen, also $l = \sup(D_{x_i})$. Es werden die Hilfsvariablen $r_2, ..., r_l$ definiert. Eine Variable r_k soll die Anzahl der Vorkommnisse der Umgebungs-ID k in allen Variablen x_i enthalten. Die Umgebung mit der ID 1 ist die einzige Umgebung der Stufe *Keine Isolation* und wird aus den oben genannten Gründen bei der Zählung ignoriert. Es wird folgendes Constraint definiert, um r_k auf die Anzahl der Vorkommnisse des Wertes k zu begrenzen:

$$\text{gcc}(\{x_1, ..., x_m\}, \{(2, r_2), ..., (l, r_l)\}) \tag{6.4.1}$$

Des Weiteren werden die binären Hilfsvariablen $s_2, ..., s_l$ mit dem Wertebereich $\{0, 1\}$ definiert. s_k soll genau dann den Wert 1 annehmen, wenn die Umgebungs-ID k einer oder mehrerer Anwendungen zugeordnet ist. Ansonsten soll $s_k = 0$ sein. Dies kann durch folgende Constraints bewerkstelligt werden, wobei $1 \leq k \leq l$ gilt:

$$\forall k \; r_k > 0 \iff s_k = 1$$

Die Anzahl an Umgebungs-IDs, denen die Anwendungen zugeordnet sind, und damit die Anzahl an Isolationsumgebungen z, ergibt sich durch Aufsummieren aller s_k:

$$z = \sum_{k=2}^{l} s_k$$

Das Constraint (6.4.1) kann auch mit Hilfe des *count*-Constraints formuliert werden.

Definition 6.4.5 (count Constraint [BCR05]) *Das Constraint hat die Form* $count(v_0, \{v_1, ..., v_n\}, VERGLEICH, u)$. $v_0, ..., v_n$ *und* u *sind Variablen.* $VERGLEICH$ *ist ein Vergleichsoperator aus der Menge* $\{=, \neq, <, \leq, >, \geq\}$. *Sei* m *die Anzahl an Variablen aus* $\{v_1, ..., v_n\}$, *die den Wert von* v_0 *angenommen haben. Das Constraint ist erfüllt, wenn* m $VERGLEICH$ u *gilt.*

Haben z. B. die Variablen $(v_1, ..., v_5)$ *die Werte* $(4, 2, 2, 2, 4)$ *angenommen und es gilt* $v_0 = 2$, *dann muss* $u = 3$ *sein, damit das Constraint erfüllt ist, wenn der Vergleichsoperator '=' gewählt wurde.*

Sei m die Anzahl an Anwendungen und l die größte ID für Isolationsumgebungen. Das Constraint (6.4.1) lässt sich durch folgende Constraints ersetzen, wobei gilt $1 \leq k \leq l$:

$$\forall k \; \text{count}(k, \{x_1, ..., x_m\}, =, r_k)$$

6.4.1.5 Bevorzugen schwacher Isolationsstufen

Es gilt in der Bewertungsfunktion zu berücksichtigen, dass für die Anwendungen keine Vorgaben zu den Isolationsstufen gemacht werden müssen. Falls zu einer Anwendung keine Vorgabe zur Isolationsstufe existiert, ist eine schwache Stufe einer starken Isolationsstufe vorzuziehen, da eine stärkere Isolation auch einen höheren Verbrauch an Ressourcen bedeutet. Um diesem Umstand Rechnung zu tragen, wird eine Gewichtung der Isolationsumgebungen eingeführt. Die IDs der Isolationsumgebungen sind bereits nach Isolationsstufen aufsteigend geordnet. Eine Gewichtung g nach den Isolationsstufen lässt sich also durch das Aufsummieren der verwendeten IDs erreichen, die in den Variablen x_i enthalten sind. Folgendes Constraint verknüpt g mit dieser Berechnung.

$$g = \sum_{i=1}^{m} x_i$$

Den Maximalwert erreicht g, wenn allen Anwendungen die Isolationsumgebung mit der größten ID l zugeordnet wird.

$$g_{\max} = m \cdot l$$

Das Maximum der Gewichtung muss nicht als Constraint in die Formulierung mit aufgenommen werden, da m und l bereits vor der Lösung des Constraint Optimization Problems bekannt sind und damit $g_{\max}$ schon vorher berechnet werden kann.

Isolationsstufe j / Anwendung i		Keine Isolation	Prozess	Virtuelle Maschine
		1	2	3
Anwendung A	1	1	0	0
Anwendung B	2	1	0	0
Anwendung C	3	0	0	1
Anwendung D	4	0	0	2

Tabelle 6.2: Beispielbelegung der Variablen x_{ij}

6.4.2 Zweidimensionaler Viewpoint

6.4.2.1 Variablen

Die Variablen dieses Viewpoint sind als $m \times n$ Matrix x_{ij} angeordnet. i entspricht der Identifikationsnummer (ID) einer Anwendung. m ist die Anzahl der Anwendungen. Es gilt $1 \leq i \leq m$. j ist die ID einer Isolationsstufe. n ist die Anzahl an Isolationsstufen. Die Stufen sind aufsteigend nach dem Isolationsgrad geordnet, d. h. die IDs beginnen bei $j = 1$ mit der Stufe *Keine Isolation* und enden bei $j = n$ mit der stärksten Isolationsstufe.

Eine Variable x_{ij} entspricht der ID der Isolationsumgebung der Stufe j in der sich die Anwendung i befindet. Die IDs der Umgebungen beginnen bei 1 und sind nur innerhalb einer Isolationsstufe eindeutig. Da sich eine Anwendung nur in genau einer Umgebung genau einer Stufe befinden kann, ist, für eine bestimmte Anwendung, x_{ij} nur für genau ein j größer gleich 1 und ansonsten 0. $x_{ij} = 0$ bedeutet, dass sich die Anwendung i nicht in einer Isolationsumgebung der Stufe j befindet.

Das Beispiel einer Belegung der Variablen x_{ij} in Tabelle 6.2 soll deren Bedeutung verdeutlichen. Die Anwendungen A und B sind im Beispiel nicht isoliert. Die Anwendung C und D befinden sich in unterschiedlichen Isolationsumgebungen der Stufe *Virtuelle Maschine* (VM). Anwendung C ist in der VM-Umgebung mit der ID 1. Anwendung D ist in der VM-Umgebung mit der ID 2.

6.4.2.2 Wertebereiche

Da es höchstens so viele Isolationsumgebungen wie Anwendungen geben kann, ist der Wertebereich von x_{ij} durch die Anzahl an Anwendungen m nach oben hin begrenzt. Außer

im Fall der Isolationsstufe *Keine Isolation*. Da es nur eine Umgebung dieser Stufe gibt, ist x_{i1} durch 1 nach oben hin begrenzt.

$$D_{x_{ij}} = \begin{cases} [0;1] \subset \mathbb{N}, & \text{wenn } j = 1 \\ [0;m] \subset \mathbb{N}, & \text{sonst} \end{cases}$$

6.4.2.3 Constraints

Ohne weitere Einschränkung der Variablenbelegung ist es möglich, dass einer Anwendung mehrere oder keine Isolationsumgebungen zugewiesen werden. Die folgenden Constraints wirken dem entgegen. Die Anzahl an Isolationsstufen wird durch n dargestellt.

$$\forall i \ \max_j(x_{ij}) = \sum_{j=1}^{n} x_{ij}$$

$$\forall i \ \sum_{j=1}^{n} x_{ij} > 0$$

Die erste Bedingung besagt, dass sich eine Anwendung in höchstens einer Isolationsumgebung befinden darf. Die zweite Bedingung stellt sicher, dass eine Anwendung in mindestens einer Isolationsumgebung ausgeführt wird. Zusammen bewirken die Constraints, dass einer Anwendung genau eine Isolationsumgebung zugeordnet wird.

6.4.2.4 Isolationsvorgaben

Isolationsvorgaben werden als Constraints formuliert. Es folgt eine Liste von Constraints für die grundlegenden Isolationsvorgaben. Für Programmbibliotheken, die keine Reified Constraints (siehe Definition 6.4.2) anbieten, werden Alternativen angegeben, die logische Ausdrücke mit Hilfe der Betragsfunktion und der Signum-Funktion definieren.

Anwendung a ist in einer anderen Isolationsumgebung als Anwendung b:

$$\forall j \ (x_{aj} \neq x_{bj}) \vee (x_{aj} = x_{bj} \wedge x_{aj} = 0 \wedge x_{bj} = 0)$$

Dieser Ausdruck lässt sich vereinfachen zu:

$$\forall j \ (x_{aj} \neq x_{bj}) \vee (x_{aj} = 0)$$

Die Alternative ohne Nutzung von logischen Verknüpfungen lautet:

$$\forall j \ \operatorname{sgn}(|x_{aj} - x_{bj}|) + |\operatorname{sgn}(x_{aj}) - 1| > 0$$

Anwendung a ist in einer Isolationsumgebung der Isolationsstufe k oder l:

$$(x_{ak} \neq 0) \vee (x_{al} \neq 0)$$

Die Alternative ohne Nutzung von logischen Verknüpfungen lautet:

$$x_{ak} + x_{al} > 0$$

Anwendung a ist in derselben Isolationsumgebung wie Anwendung b:

$$\forall j \; x_{aj} = x_{bj}$$

Es gilt zu beachten, dass sich die alternativen Constraints den Umstand zunutze machen, dass x_{ij} aufgrund des Wertebereichs immer positiv ist ($\forall i \forall j \; x_{ij} \geq 0$).

6.4.2.5 Anzahl an Isolationsumgebungen

Die zu minimierende Bewertungsfunktion besteht u.a. aus der Anzahl an Isolationsumgebung z. Die einzige Umgebung der Stufe *Keine Isolation* ($j = 1$), das Haupt-Framework, existiert auch, wenn keine Anwendung darin ausgeführt wird. Es verursacht also keinen zusätzlichen Ressourcenverbrauch, eine Anwendung in dieser Stufe auszuführen. Deshalb wird die Umgebung der Stufe auch nicht in der Anzahl der Isolationsumgebungen berücksichtigt.

Sei n die Anzahl an Isolationsstufen. Eine Näherung der Anzahl an Isolationsumgebungen erhält man folgendermaßen:

$$z = \sum_{j=2}^{n} \left(\max_{i}(x_{ij}) \right)$$

Es werden die höchsten Umgebungs-IDs der Isolationsstufen zusammengezählt. Für den Fall, dass die Umgebungs-IDs jeder Stufe ein lückenloses Intervall beginnend bei 1 ergeben, erhält man genau die Anzahl an Isolationsumgebungen. Ist dies nicht der Fall, erhält man eine Höhere als die tatsächliche Anzahl. Das beeinträchtigt jedoch nicht die Korrektheit der optimalen Lösungen. Denn die Bewertungsfunktion wird minimiert, womit die Lösungen mit höheren Anzahlen als schlecht bewertet werden und nicht in der optimalen Lösungsmenge auftauchen.

6.4.2.6 Bevorzugen schwacher Isolationsstufen

Falls aus den Isolationsvorgaben keine eindeutige Zuordnung einer Anwendung zur einer Isolationsstufe abgeleitet werden kann, besteht die Wahlmöglichkeit zwischen unterschiedlichen Stufen. Dabei ist eine schwache Stufe einer starken Isolationsstufe vorzuziehen. Um

diesen Umstand in die Bewertungsfunktion einfließen zu lassen, ist eine Gewichtung der Isolationsumgebungen nötig.

Im Folgenden ist m die Anzahl an Anwendungen und n die Anzahl an Isolationsstufen.

Die Bewertungsfunktion soll minimiert werden, also muss den, zu bevorzugenden, Umgebungen eine niedrige Gewichtung zuteil werden. Das heißt, es müssen alle Umgebungen einer schwachen Isolationsstufe ein niedrigeres Gewicht erhalten als Umgebungen einer stärkeren Stufe. Das lässt sich durch folgende Gewichtung h_{ij} für jede Variable x_{ij} erreichen:

$$\forall i \forall j \; h_{ij} = x_{ij} \cdot m^{j-1}$$

Die Gewichtung g der gesamten Lösung erhält man durch folgende Berechnung, die als Constraint in die Formulierung des Constraint Optimization Problems aufgenommen wird:

$$g = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} h_{ij} \right)$$

Das Maximum der Gewichtung ist erreicht, wenn alle Komponenten der Matrix x_{ij} ihren Höchstwert annehmen. Für $j = 1$ ist der Höchstwert $\max(x_{i1}) = 1$, da für alle i gilt: $\sup(D_{x_{i1}}) = 1$. Für $j > 1$ ist der Höchstwert $\max(x_{ij}) = m$.

$$g_{\max} = m \cdot 1 + \sum_{j=2}^{n} \left(m^2 \cdot m^{j-1} \right) = m + \sum_{j=3}^{n+1} m^j$$

Diese Berechnung kann bereits vor der Lösung des Constraint Optimization Problem durchgeführt werden, da die Anzahl an Anwendungen m und Isolationsstufen n schon vorher bekannt sind.

Mit Hilfe der Berechnungsformel für geometrische Reihen lässt sich die Summenformel in $g_{\max}$ auflösen:

$$g_{\max} = \frac{m^{n+2} - 1}{m - 1} - m^2 - 1$$

6.4.3 Bewertungsfunktion

Die zu minimierende Bewertungsfunktion o setzt sich zusammen aus der Anzahl der Isolationsumgebungen z und der Gewichtung g. Wie diese Werte für die unterschiedlichen Viewpoints berechnet werden, wurde bereits in den vorangegangenen Abschnitten beschrieben (siehe 6.4.1, 6.4.2).

$$o = z \cdot g_{\max} + g \tag{6.4.2}$$

Diese Bewertungsfunktion gibt einer niedrigen Anzahl an Isolationsumgebungen die absolute Priorität.

Die folgende Bewertungsfunktion stellt einen Kompromiss zwischen niedriger Anzahl an Umgebungen und möglichst schwacher Isolation zu Gunsten einer besseren Leistung für die Anwendungen dar:

$$o = g \tag{6.4.3}$$

Hierbei wird die Anzahl an Isolationsumgebungen nur innerhalb einer Isolationsstufe minimiert. Das bedeutet, dass jede Anwendung in einer Umgebung einer möglichst schwachen Stufe ausgeführt wird, auch wenn dazu eine zusätzliche Umgebung erzeugt werden muss.

Der Unterschied zwischen den beiden Varianten wird anhand eines einfachen Beispiels klar: Betrachtet werden vier Anwendungen *a*, *b*, *c* und *d*. Zur Verfügung stehen die Isolationsstufen *Keine Isolation*, *Prozess* und *Virtuelle Maschine* (VM) (aufsteigend geordnet nach der Stärke der Isolation). Es werden die folgenden Isolationsvorgaben gestellt:

- *a* soll in der Isolationsstufe VM ausgeführt werden.
- *b*, *c* und *d* sollen in einer stärkeren Isolationsstufe als der Stufe *Keine Isolation* ausgeführt werden.

Die Verwendung der Bewertungsfunktion (6.4.2) hat zur Folge, dass alle Anwendungen in einer Virtuellen Maschine ausgeführt werden. Das heißt, *b*, *c* und *d* werden in einer VM ausgeführt, obwohl eine so starke Isolation für die Anwendungen gar nicht gefordert wurde. Dafür gibt es aber auch nur eine Isolationsumgebung.

Die Bewertungsfunktion (6.4.3) zieht die Lösung vor, *a* in einer VM auszuführen und *b*, *c*, *d* gemeinsam in einem Prozess. Es werden demnach zwei Isolationsumgebungen benötigt. Also auch mehr Ressourcen des Wirtsystems verbraucht als mit der Lösung der anderen Funktion. Allerdings werden die Anwendungen *b*, *c* und *d* nicht durch die Vorgabe für *a* „ausgebremst“ und auch in einer VM ausgeführt.

6.4.4 Statische Einschränkung der Wertebereiche

Je kleiner der Wertebereich der Variablen eines Constraint Optimization Problems, desto weniger Belegungen und damit Lösungen sind möglich. Also schränkt eine Verkleinerung des Wertebereichs den Suchraum eines Problems ein, was zu einer verbesserten Laufzeit führen kann. Die folgenden Verfahren machen sich diesen Umstand zunutze, indem die Wertebereiche vor der Lösungssuche angepasst werden. Es werden Werte ausgeschlossen, von denen bereits vor der Lösungssuche bekannt ist, dass sie zu ungültige Belegungen oder nicht optimalen Lösungen führen. Da nur die Problemformulierung angepasst wird, nicht jedoch die Programmbibliothek zur Lösungssuche, lassen sich diese Verfahren unabhängig von der Programmbibliothek umsetzen.

6.4.4.1 Einschränkung auf Basis der Isolationsvorgaben

Bei der Auflösung von Isolationsvorgaben sind die Werte, in beiden Viewpoints, Identifikationsnummern (IDs) für Isolationsumgebungen. Für unterschiedliche Isolationsstufen existieren auch unterschiedliche IDs. Wenn schon vor der Auflösung der Isolationsvorgaben bekannt ist, dass Umgebungen einer bestimmten Isolationsstufe nicht benötigt werden, kann dieses Wissen genutzt werden um die Wertebereiche einzuschränken.

Wenn für alle Anwendungen genau eine Isolationsstufe vorgegeben wird, können die IDs der ungenutzten Stufen aus den Wertebereichen entfernt werden. Da für jede Stufe bekannt ist, welche Anzahl an Anwendungen der Stufe zugeordnet sind, kann der Wertebereich der benutzten Stufen ebenfalls reduziert werden. In beiden Viewpoints werden, bei einer Anzahl von m Anwendungen, jeweils m Umgebungs-IDs für *jede* Isolationsstufe vorgehalten, obwohl insgesamt nur m IDs vergeben werden können. Mit dem Wissen über die Anzahl an Anwendungen pro Stufe kann die Obergrenze von m auf die jeweilige Anzahl abgesenkt werden.

6.4.4.2 Iterative Vergrößerung der Wertebereiche

Die beschriebene Verkleinerung des Wertebereichs funktioniert nur, wenn für alle Anwendungen Isolationsstufen vorgegeben werden. Mit einem iterativen Verfahren ist es jedoch auch möglich, den Wertebereich zu verkleinern, ohne dass Isolationsstufen vorgegeben werden müssen. Dazu wird die Programmbibliothek zur Lösung des Constraint Optimization Problems mehrfach, mit wachsendem Wertebereich, aufgerufen. Man startet mit einem Wertebereich der nur eine ID für die Stufe *Keine Isolation* vorhält. Führt dieser Aufruf zu keiner Lösung wird der Wertebereich schrittweise vergrößert, beginnend bei der schwächsten Isolationsstufe. Nach jeder Vergrößerung wird wieder die Programmbibliothek aufgerufen. Ist die maximale Anzahl m an IDs einer Isolationsstufe erreicht, wird der Wertebereich der IDs der nächsten Stufe schrittweise vergrößert, bis eine Lösung gefunden wird. Falls die Isolationsvorgaben nicht erfüllbar sind, wird der Wertebereich schrittweise vergrößert, bis die letzte ID der stärksten Isolationsstufe erreicht ist.

Voraussetzung für die korrekte Funktionsweise dieses Verfahrens ist, dass die Lösungssuche bei Vorgaben, die aufgrund eines verkleinerten Wertebereichs unerfüllbar sind, schnell beendet ist. Ansonsten dauern die Iterationen, in denen der Wertebereich schrittweise vergrößert wird, möglicherweise länger als die Lösungssuche ohne das Verfahren.

6.4.4.3 Kombination beider Verfahren

Die oben beschriebenen Verfahren zur Einschränkung der Wertebereiche lassen sich kombinieren. Zunächst wird die Einschränkung auf Basis der Isolationsvorgaben durchgeführt und anschließend die iterative Vergrößerung.

Die Einschränkung auf Basis der Isolationsvorgaben wird in abgewandelter Form verwendet. Es werden zunächst die Isolationsstufen identifiziert, die in Vorgaben vorkommen, die Anwendungen auf eine oder mehrere Stufen beschränken. Wird eine Anwendung auf mehrere Stufen beschränkt, wird bei der Einschränkung der Wertebereiche nur die niedrigste Stufe berücksichtigt. Die Umgebungs-IDs der identifizierten Stufen werden, bis auf eine, aus den Wertebereichen entfernt. Die IDs der anderen Stufen werden vollständig aus den Wertebereichen entfernt. Eine Ausnahme bildet die ID der Umgebung der Stufe *Keine Isolation*, die immer vorgehalten wird. Dann wird die Lösungssuche durchgeführt. Falls sie keine Lösung liefert, wird den Wertebereichen der identifizierten Stufen jeweils eine ID hinzugefügt und die Lösungssuche wiederholt. Einer identifizierten Stufe werden schrittweise IDs hinzugefügt, bis die Anzahl der Anwendungen, die auf diese Stufe beschränkt wurden, der Anzahl an IDs der Stufe entspricht oder eine Lösung gefunden wird. Wird für alle Stufen diese Grenze erreicht, ohne dass eine Lösung gefunden wird, beginnt die iterative Vergrößerung wie im vorigen Abschnitt 6.4.4.2 beschrieben.

6.4.5 Symmetry Breaking

In einem Constraint Satisfaction Problem (CSP) können Symmetrien auftreten, die den Suchraum vergrößern und damit die Laufzeit verlängern können. Durch Symmetrien entstehen zusätzliche gleichwertige Lösungen, die sich nur durch einen, für das eigentliche Problem irrelevanten, Aspekt unterscheiden. Beispielsweise existieren bei der Lösung eines Knotenfärbungsproblems Symmetrien, wenn ein Knoten als Variable des CSP und eine Farbe als Wert der Variablen dargestellt wird. Es entstehen Lösungen, die sich nur dadurch von anderen Lösungen unterscheiden, dass zwei Farben miteinander vertauscht wurden. Jedoch ist bei einer Knotenfärbung nicht der Farbwert maßgebend. Interessant ist vielmehr, welchen Knoten die gleiche Farbe zugeordnet wurde.

Ziel des *Symmetry Breaking* ist das Aufbrechen von Symmetrien, so dass möglichst nur noch eine Lösung jeder Äquivalenzklasse im Suchraum bestehen bleibt.

Symmetrien in Constraint Satisfaction Problems lassen sich in zwei grundlegende Arten unterteilen [Walo7]. In beiden Fällen geht es darum, in einer Variablenbelegung, die eine Lösung darstellt, Vertauschungen durchzuführen, um eine weitere Lösung zu erhalten. Die Constraints des Problems bleiben bei der Permutation unbeachtet. Bei einer *Variablensymmetrie* werden die Variablen in der Lösung vertauscht. Eine *Wertesymmetrie* führt eine Permutation der Werte durch. Im angeführten Beispiel des Knotenfärbungsproblems liegt, durch die Möglichkeit des Vertauschens von Farben, eine Wertesymmetrie vor. Zudem handelt es sich bei den Farben um *ununterscheidbare Werte*.

In den folgenden Definitionen von Symmetrien, die von Walsh [Walo7] stammen, werden Variablen mit x_i bezeichnet und Werte mit d_i. Z ist die Menge aller Variablen eines Constraint Satisfaction Problems. Alle Variablen haben den gleichen Wertebereich D. Dies kann ohne

Beschränkung der Allgemeinheit angenommen werden, da Wertebereiche einzelner Variablen durch Constraints individuell angepasst werden können.

Definition 6.4.6 (Variablensymmetrie [Wal07]) *Eine Variablensymmetrie (engl.* variable symmetry*) ist eine Bijektion σ, die auf die Indizes der Variablen eines Constraint Satisfaction Problems angewendet wird, so dass gilt: Wenn $x_1 = d_1, ..., x_n = d_n$ eine Lösung ist, dann ist auch $x_{\sigma(1)} = d_1, ..., x_{\sigma(n)} = d_n$ eine Lösung.*

Definition 6.4.7 (Wertesymmetrie [LL06]) *Eine Wertesymmetrie (engl.* value symmetry*), über den Variablen $U \subseteq Z$ eines Constraint Satisfaction Problems, ist eine Bijektion $\theta : D \to D$, die auf eine Lösung angewendet wird, so dass gilt: Wenn $\{x_i = d_i \mid x_i \in Z\}$ eine Lösung ist, dann ist auch $\{x_i = d_i \mid x_i \in Z \setminus U\} \cup \{x_i = \theta(d_i) \mid x_i \in U\}$ eine Lösung.*

Definition 6.4.8 (Ununterscheidbare Werte [LL06]) *Für eine Menge von Werten $F \subseteq D$ mit k Elementen können $k!$ verschiedene Bijektionen $\kappa_l : F \to F$ definiert werden, wobei gilt $1 \leq l \leq k!$. Handelt es bei jeder dieser Bijektionen κ_l um eine Wertesymmetrie über den Variablen $U \subseteq Z$, dann werden die Werte aus F* ununterscheidbare Werte *(engl.* indistinguishable values*) über den Variablen U genannt.*

Beide Symmetrien können gleichzeitig vorkommen. Zum Beispiel kann im angeführten Knotenfärbungsproblem auch eine Variablensymmetrie auftreten, wenn der zu färbende Graph $G = (V, E)$ Knoten enthält, die die selben Nachbarn haben. Wenn also für zwei Knoten u_1, u_2 und alle Knoten $v \in V$ gilt: $(u_1, v) \in E \Leftrightarrow (u_2, v) \in E$, dann existiert eine Variablensymmetrie für die Variablen der Knoten u_1 und u_2.

Die Definitionen lassen sich auch auf Constraint Optimization Problems (COPs) anwenden [Wal07]. Dabei ist jedoch zu beachten, dass eine symmetrische Lösung nicht zwingend den gleichen Wert der Bewertungsfunktion erhält. Beim Entfernen von Symmetrien in COPs muss also darauf geachtet werden, dass keine optimale Lösung zu Gunsten ihres symmetrischen Pendants ausgeschlossen wird.

6.4.5.1 Wertesymmetrie bei der Auflösung von Isolationsvorgaben

Beide Viewpoints enthalten eine oder mehrere Variablen für eine Anwendung. Die Werte der Variablen stellen Identifikationsnummern (IDs) für die Isolationsumgebungen dar. Für die Isolationsumgebungen einer Isolationsstufe existiert, unabhängig von den Isolationsvorgaben, eine Wertesymmetrie. Das gilt für alle Stufen außer der Stufe *Keine Isolation*, der immer nur eine Isolationsumgebung angehört.

Die Symmetrie ist im Beispiel in Tabelle 6.2 gut erkennbar. Den Variablen $x_{3,3}$ und $x_{4,3}$ sind die Werte 1 bzw. 2 zugeordnet. Die Werte dieser Variablen können einfach miteinander vertauscht werden, um eine äquivalente Lösung zu erhalten. Denn die Aussage dieser Belegung ist, dass den Anwendungen C und D unterschiedliche Isolationsumgebungen

zugeordnet sind. Der eigentliche Zahlenwert der IDs ist dabei irrelevant. Relevant ist nur, wie die Anwendungen gruppiert sind. Im Beispiel wurde zudem der Wertebereich $D_{x_{i,3}} = [0, 4]$ nicht ausgenutzt. Es können also zusätzlich die IDs 3 und 4 gegen die Werte von $x_{3,3}$, $x_{4,3}$ ausgetauscht werden, solange die Bedingung $x_{3,3} \neq x_{4,3}$ gewahrt bleibt. Der Wert 0 stellt keine ID dar, sondern bedeutet, dass die Anwendung sich nicht in der jeweiligen Isolationsstufe befindet. Deshalb kann er nicht ausgetauscht werden oder einen anderen Wert ersetzen.

Im Allgemeinen können alle Werte, bis auf die 0, einer Spalte j der Matrix x_{ij} ausgetauscht werden, solange die Gruppierung der Werte erhalten bleibt. Das heißt, wenn vor dem Austausch $x_{kj} = x_{lj}$ für beliebiges k und l gilt, dann muss auch danach $x_{kj} = x_{lj}$ gelten. Gilt vor dem Austausch $x_{kj} \neq x_{lj}$, dann muss dies auch danach gelten.

Die formale Definition der Wertesymmetrien des zweidimensionalen Viewpoints lautet also wie folgt. Für die Variablen x_{ij} einer Isolationsstufe $j > 1$ existiert eine Wertesymmetrie in Form der Bijektion θ_j:

$$\theta_j(d) = \begin{cases} 0, & \text{wenn } d = 0 \\ e \in D_{x_{ij}} \setminus \{0\} \mid \forall k \forall l \; x_{kj} = x_{lj} \Leftrightarrow \theta_j(x_{kj}) = \theta_j(x_{lj}), & \text{sonst} \end{cases}$$

Die Werte $D_{x_{ij}} \setminus \{0\}$ einer Isolationsstufe $j > 1$ sind demnach ununterscheidbare Werte über den Variablen x_{ij}.

Für die Stufe $j = 1$ (Keine Isolation) existiert nur eine Isolationsumgebung. Es gibt also auch nur genau eine ID in dieser Stufe. Es sind also keine Permutationen möglich, weshalb für die Variablen x_{i1} dieser Stufe auch keine Wertesymmetrien existieren.

Für den eindimensionalen Viewpoint existiert prinzipiell die gleiche Symmetrie. Da der Wertebereich der Variablen jedoch in Teilintervalle für die einzelnen Isolationsstufen aufgeteilt ist, erzeugen nur Permutationen der Werte innerhalb der Teilintervalle äquivalente Lösungen. Denn vertauscht man den Wert einer Variable mit einem Wert der nicht im gleichen Teilintervall liegt, verschiebt man die zugehörige Anwendung in eine Umgebung einer anderen Isolationsstufe. Auch bei diesem Viewpoint gilt, dass nur die Gruppierung der Anwendungen in Isolationsumgebungen relevant ist, nicht der Zahlenwert der IDs, sofern die Grenzen der Teilintervalle nicht übertreten werden.

Die Wertesymmetrie θ des eindimensionalen Viewpoints ist für die Variablen $x_1, ..., x_m$ folgendermaßen definiert. ID_j beschreibt das Teilintervall der Stufe j im Wertebereich der Variablen.

$$\begin{aligned} &\theta(d) \in \mathrm{ID}_j \mid d \in \mathrm{ID}_j \\ \wedge \; &\forall k \forall l \; x_k = x_l \Leftrightarrow \theta(x_k) = \theta(x_l) \end{aligned}$$

Folglich handelt es sich bei den Werten ID_j einer Isolationsstufe j um ununterscheidbare Werte über allen Variablen $x_1, ..., x_m$.

6.4.5.2 Variablensymmetrie bei der Auflösung von Isolationsvorgaben

Prinzipiell besteht zwischen den Variablen zweier Anwendungen eine Wertesymmetrie, wenn die Anwendungen derselben Isolationsstufe zugewiesen wurden und ein Vorgabe zum Ausschluss der Nachbarschaft zwischen den Anwendungen existiert. Kommen jedoch weitere Vorgaben hinzu, ist die Aufdeckung der Variablensymmetrien nicht mehr trivial, sondern mit hohem Aufwand verbunden. Deshalb beschränkt sich die Arbeit im Weiteren auf die Entfernung von Wertesymmetrien. Trotzdem soll noch ein Verfahren zur Entfernung von Variablensymmetrien vorgestellt werden, da es auch bei der Entfernung von Wertesymmetrien hilfreich sein kann (siehe 6.4.5.4).

Lex-Leader Constraints

Im Idealfall entfernt das Symmetry Breaking alle Belegungen aus einer Äquivalenzklasse bis auf einen Repräsentanten. Zur Entfernung von Variablensymmetrien existiert ein einfaches Verfahren, das einen solchen Repräsentanten bestimmt. Mit sogenannten *Lex-Leader* Constraints wird eine lexikographische Ordnung auf den betroffenen Variablen definiert [RBW06]. Die Constraints werden einfach dem bestehenden Constraint Satisfaction Problem hinzugefügt. Entspricht eine Variablenbelegung nicht dieser Ordnung, wird sie von der Lösungsmenge ausgeschlossen. Dadurch bleibt nur eine Lösung aus jeder Äquivalenzklasse übrig. Und zwar die Lösung, die der lexikographischen Ordnung entspricht.

Zur Definition von Lex-Leader Constraints wird die Relation $\leq_{lex}$ verwendet. Sie definiert eine lexikographische Ordnung über Tupeln von Variablen, genauer über deren Belegungen. Mit Hilfe der totalen Ordnung $\leq$ über den natürlichen Zahlen kann eine lexikographische Ordnung über Tupeln von natürlichen Zahlen definiert werden. Diese Ordnung kann eingesetzt werden, um Variablenbelegungen miteinander zu vergleichen.

Das Lex-Leader Constraint $(x_1, x_2) \leq_{lex} (x_3, x_4)$ gilt, wenn für die Belegung der Variablen gilt $(x_1 \leq x_3)$ oder $(x_1 = x_3 \wedge x_2 \leq x_4)$.

Folgendes Beispiel demonstriert die Verwendung von Lex-Leader Constraints. Es sei eine Variablensymmetrie zwischen den Spalten der Matrix:

$$\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix}$$

Die Variablensymmetrie σ ist also: $\sigma(1) = 2$, $\sigma(3) = 4$, $\sigma(2) = 1$, $\sigma(4) = 3$. Zur Entfernung dieser Symmetrie kann folgendes Constraint definiert werden:

$$(x_1, x_3) \leq_{lex} (x_2, x_4)$$

Das Verfahren kann auch auf Constraint Optimization Problems angewendet werden. Jedoch muss, wie bei allen Verfahren zum Symmetry Breaking, darauf geachtet werden, dass keine Belegung ausgeschlossen wird, die Teil einer optimalen Lösung ist.

6.4.5.3 Dynamische Einschränkung der Wertebereiche

Van Hentenryck beschreibt in [VH02] eine Anpassung des Suchalgorithmus für das Scene Allocation Problem, dass eine ähnliche Wertesymmetrie aufweist, wie das vorliegende Problem. Bei der Scene Allocation müssen Filmszenen, die von bestimmten Schauspielern gespielt werden, auf Tage verteilt werden. Es können jeweils nur fünf Szenen pro Tag gedreht werden. Die Schauspieler werden pro Tag bezahlt. Das Ziel sind möglichst geringe Produktionskosten. Es geht also darum, die Szenen so zu gruppieren, dass jeder Schauspieler möglichst wenige Tage beschäftigt ist. Für jede Szene existiert eine Variable, die mit dem zugeordneten Tag belegt wird. Es herrscht also eine Wertesymmetrie, da die Werte der Tage beliebig ausgetauscht werden können, solange die Gruppierung der Szenen erhalten bleibt.

Van Hentenryck löst die Symmetrie durch eine dynamische Einschränkung der Wertebereiche der Variablen. Bei der Lösungssuche von Constraint Optimization Problems werden die Variablen sukzessiv mit Werten belegt. Die dynamische Einschränkung legt den Wertebereich einer Variable während der Lösungssuche, aufgrund der Werte, die bereits anderen Variablen zugewiesenen wurden, fest.

Für die erste Szene wird festgelegt, dass sie am ersten Tag gedreht wird. Szene 2 kann am selben Tag oder einem beliebigen anderen Tag gedreht werden. Da es bei diesem Problem keinen Unterschied macht, welcher der anderen Tage gewählt wird, wird ein Wert festgelegt, um Symmetrien zu vermeiden. Van Hentenryck wählt einfach den nächsten Tag: Tag 2. Bei der Lösungssuche müssen dann nur noch diese beiden Tage für Szene 2 berücksichtigt werden. Für die Planung der weiteren Szenen gilt das gleiche Prinzip. Es müssen nur die bereits min. einer Szene zugeordneten Tage, sowie ein noch völlig freier Tag berücksichtigt werden. Als freier Tag wird immer der kleinste noch nicht vergebene Wert gewählt.

Für die Auflösung der Isolationsvorgaben mit Hilfe des eindimensionalen Viewpoints (siehe 6.4.1) kann eine ähnliche Einschränkung des Wertebereichs verwendet werden. Die Szenen des Scene Allocation Problem entsprechen den Anwendungen. Die Tage entsprechen den Isolationsumgebungen. Neben den bereits zugeordneten Umgebungen wird bei der Lösungssuche die freie Umgebung mit der kleinsten ID berücksichtigt. Zusätzlich müssen im Allgemeinen jedoch noch weitere Umgebungen in Betracht gezogen werden. Denn nicht alle Isolationsumgebungen sind als gleichartig anzusehen. Sie gehören unterschiedlichen Isolationsstufen an. Es muss also, zusätzlich zu den bereits zugeordneten Isolationsumgebungen, eine freie Umgebung aus jeder Isolationsstufe berücksichtigt werden. Aufgrund der Unterscheidung in Isolationsstufen kann der ersten Anwendung auch nicht einfach die Umgebung mit der ID 1 zugewiesen werden. Denn möglicherweise kommt die zugehörige Stufe,

aufgrund der Isolationsvorgaben, für die Anwendung nicht in Frage. Deswegen wird auch für die erste Anwendung aus jeder Isolationsstufe eine Umgebungen bei der Lösungssuche berücksichtigt.

Im zweidimensionalen Viewpoint (siehe 6.4.2) sind die Isolationsstufen auf die Spalten j der Variablenmatrix x_{ij} aufgeteilt. Einer Anwendung gehört für jede Isolationsstufe jeweils eine Variable an. Die Stufen spiegeln sich also nicht mehr direkt in den Werten wieder. Deshalb müssen sie bei der Einschränkung der Wertebereiche nicht mehr berücksichtigt werden. Für die Variable x_{ij} einer Anwendung i müssen nur die Werte der bereits belegten Variablen der Spalte j, sowie ein, bisher in der Spalte ungenutzter, Wert bei der Lösungssuche berücksichtigt werden.

6.4.5.4 0/1-Viewpoint

Zur Entfernung von Variablensymmetrien wurde bereits das Lex-Leader-Verfahren vorgestellt. Es muss lediglich ein zusätzliches Constraint hinzugefügt werden, um den Symmetrien entgegenzuwirken. Um dieses einfache Verfahren auf eine Wertesymmetrie anwenden zu können, muss die Wertesymmetrie auf eine Variablensymmetrie abgebildet werden. Im Folgenden wird beschrieben, wie diese Abbildung durchgeführt werden kann, um die Wertesymmetrien bei der Auflösung von Isolationsvorgaben mit dem Lex-Leader-Verfahren zu entfernen.

Bei dem Problem der Auflösung von Isolationsvorgaben handelt es sich um ein *Multi-Aspect Assignment Problem*.

> **Definition 6.4.9 (Multi-Aspect Assignment Problem [LL06])** *Ein Aspekt eines Multi-Aspect Assignment Problem (MAP) ist ein Objekttyp. Ein MAP besteht aus Objekten unterschiedlicher Typen/Aspekte. O.B.d.A. kann angenommen werden, dass die Objekte eines Typs t, durch natürliche Zahlen aus dem Intervall $O_t = [1; m_t]$ dargestellt werden, wobei m_t die Anzahl der Objekte vom Typ t ist. Eine Lösung L eines MAP mit n Aspekten ist eine oder mehrere Kombinationen $(o_1, ..., o_n) \mid o_i \in O_i$ aus Objekten unterschiedlichen Typs, die die Constraints des Problems erfüllen.*

Im vorliegenden Problem existieren drei Aspekte: Anwendungen O_1, Isolationsstufen O_2 und Isolationsumgebungen O_2. Eine Lösung L ordnet jeder Anwendung eine Isolationsumgebung einer bestimmten Isolationsstufe zu: $L \subset (O_1 \times O_2 \times O_3)$. Da für jede Anwendung nur eine Zuordnung möglich ist, gilt: $|L| = |O_1|$.

Für ein Multi-Aspect Assignment Problem kann ein Constraint Satisfaction Problem gebildet werden, dessen Variablen und Wertebereiche durch einen *Aspect-Viewpoint* dargestellt werden.

Isolationsstufe j / Anwendung i	1	2	3
1	1	0	0
2	0	0	1
3	0	0	2
4	0	0	3

Tabelle 6.3: Aspect-Viewpoint x_{ij}

Isolationsstufe j	1	2				3			
Isolationsumgebung k / Anwendung i	1	1	2	3	4	1	2	3	4
1	1	0	0	0	0	0	0	0	0
2	1	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	1	0

Tabelle 6.4: 0/1-Viewpoint z_{ijk}

Definition 6.4.10 (Aspect-Viewpoint [LL06]) *Zu einem Multi-Aspect Assignment Problem (MAP) mit n Aspekten kann ein Aspect-Viewpoint gebildet werden, dessen Variablen als Matrix mit* $(n-1)$ *Dimensionen angeordnet sind.* $(n-1)$ *Aspekte des MAP bilden die Indizes der Variablen. Die Objekte eines Aspekts bilden den Wertebereich der Variablen. Seien die Objekttypen* $1, ..., n-1$ *die Typen zur Bildung der Indizes und Objekttyp n der Typ zur Bildung des Wertebereichs. Dann sind die Variablen definiert durch:* $x_{i_1,...,i_{n-1}} \mid i_j \in O_j$*. Der Wertebereich der Variablen ist die Menge* O_n*.*

Der zweidimensionale Viewpoint stellt einen Aspect-Viewpoint für das Problem zur Auflösung von Isolationsvorgaben dar. Die Aspekte *Anwendungen* und *Isolationsstufen* bilden die Indizes der zweidimensionalen Matrix x_{ij}. Der Wertebereich der Variablen wird durch die Isolationsumgebungen gebildet.

Neben den Aspect-Viewpoints kann auch ein 0/1-Viewpoint für ein Multi-Aspect Assignment Problem gebildet werden.

Definition 6.4.11 (0/1-Viewpoint [LL06]) *Zu einem Multi-Aspect Assignment Problem (MAP) mit n Aspekten kann ein 0/1-Viewpoint gebildet werden, dessen Variablen als n-dimensionale Matrix angeordnet sind. Alle n Aspekte des MAP bilden die Indizes der Variablen. Der Wertebereiche der Variablen besteht nur aus den Zahlen* 0 *und* 1*. Eine Variable* $x_{i_1,...,i_n} \mid i_j \in O_j$ *ist genau dann* 1*, wenn das Tupel* $(i_1, ..., i_n)$ *Teil einer Lösung L des Problems ist.*

In den Tabellen 6.3 und 6.4 wird eine Belegungen des Aspect-Viewpoints mit der entsprechenden Belegung des 0/1-Viewpoints verglichen.

Die Wertesymmetrien eines Aspect-Viewpoint eines MAP können immer auf Variablensymmetrien des 0/1-Viewpoint abgebildet werden [LL06].

Auf Basis des 0/1-Viewpoints kann also ein Constraint Optimization Problem zur Auflösung von Isolationsvorgaben formuliert werden, das, statt den Wertesymmetrien des Aspect-Viewpoints, Variablensymmetrien besitzt. Diese können mit Hilfe des Lex-Leader-Verfahrens gelöst werden.

Jedoch müssten dann auch andere Constraints für die Isolationsvorgaben formuliert werden, sowie eine angepasste Bewertungsfunktion entwickelt werden. Um diesen Aufwand zu vermeiden, kann der 0/1-Viewpoint einfach einer bestehenden Formulierung hinzugefügt werden. Der Aspect-Viewpoint wird mit dem 0/1-Viewpoint verknüpft, so dass die Belegungen der Viewpoints immer die gleiche Lösung repräsentieren. Das bedeutet, dass Constraints, die für einen Viewpoint definiert werden, sich auch auf den Anderen auswirken. Entfernt man also die Variablensymmetrien im 0/1-Viewpoint, werden auch die entsprechenden Wertesymmetrien im Aspect-Viewpoint entfernt. Diese Technik, mehrere Viewpoints zu definieren und miteinander zu verknüpfen, wird *Redundant Modeling* genannt [CLW96]. Die Constraints zur Verknüpfung der Viewpoints werden als *Channeling Constraints* bezeichnet [CLW96].

Die Matrix x_{ij} des zweidimensionalen (Aspect-)Viewpoints 6.4.2 kann mit der dreidimensionale Matrix z_{ijk} des 0/1-Viewpoints verknüpft werden. Dazu werden die folgenden Channeling Constraints definiert:

$$\forall i \forall j \forall k \; z_{ijk} = 1 \Leftrightarrow x_{ij} = k$$

Die Wertesymmetrie, die durch die ununterscheidbaren Werte der Umgebungs-IDs einer Isolationsstufe $j > 1$ hervorgerufen werden, entsprechen der Variablensymmetrie, die zwischen den Tupeln $(z_{1j1}, ..., z_{mj1}), (z_{1j2}, ..., z_{mj2}), ..., (z_{1jm}, ..., z_{mjm})$ herrscht, wobei m die Anzahl an Anwendungen ist. Diese Tupel entsprechen den Spalten in Abbildung 6.4, wobei jeweils nur zwischen den Spalten einer Isolationsstufe $j > 1$ eine Variablensymmetrie herrscht.

Mit Hilfe der folgenden Lex-Leader Constraints kann den Wertesymmetrien des Aspect-Viewpoint x_{ij} entgegengewirkt werden, wobei n die Anzahl an Isolationsstufen ist und m die Anzahl an Anwendungen.

$$\bigwedge_{2 \leq j \leq n} \left(\bigwedge_{1 \leq k \leq m-1} (z_{1,j,k+1}, ..., z_{m,j,k+1}) \leq_{lex} (z_{1,j,k}, ..., z_{m,j,k}) \right)$$

Für den eindimensionalen Viewpoint (siehe 6.4.1) können ebenfalls Channeling Constraints definiert werden, die den 0/1-Viewpoint z_{ijk} einbinden, wobei ID_j das Intervall der Umgebung-IDs der Stufe j beschreibt.

$$\forall i \forall j \forall k \; z_{ijk} = 1 \Leftrightarrow x_i = \inf(\mathrm{ID}_j) - 1 + k$$

Damit können die obigen Lex-Leader Constraints auch für den eindimensionalen Viewpoint verwendet werden.

6.4.6 Implementierung der Constraint Optimization Problems

Die Formulierungen der Constraint Optimization Problems zur Auflösung von Isolationsvorgaben wurden mit Hilfe unterschiedlicher Programmbibliotheken implementiert. Es kamen die Programmbibliotheken Cream 1.0.6 [Tam] und Choco 2.1.0 [cho] zum Einsatz. Beide Bibliotheken stellen Constraint Solver zur Verfügung, die Constraint Optimization Problems lösen können. Dazu wird mit Hilfe der Programmierschnittstelle des jeweiligen Solvers das Problem definiert, also die Variablen, Wertebereiche, Constraints und die Bewertungsfunktion festgelegt. Dann wird das definierte Problem an den Solver übergeben, der die Lösungssuche durchführt und schlussendlich eine oder mehrere Belegungen ausgibt, die die optimalen Lösungen darstellen.

6.4.6.1 Choco

Choco ist eine umfangreiche Java-Programmbibliothek zur Lösung von Constraint Satisfaction/Optimization Problems. Es werden über 50 verschiedene Constraint-Typen angeboten. Zudem ermöglicht Choco dem Benutzer, mit Hilfe von Schnittstellen, die Lösungssuche anzupassen. Choco wird u.a. an der *École des Mines de Nantes* (Frankreich) als Open-Source-Projekt entwickelt [Choo8]. Das Projekt wurde zur Schriftlegung des vorliegenden Dokuments aktiv weiterentwickelt. Die beteiligten Entwickler bieten zudem Unterstützung bei der Nutzung der Programmbibliothek in einem Internet-Forum an. 2008 nahm Choco an der *International CSP Solver Competition* teil und belegte Top-10-Platzierungen [Choo8, CSPo8].

6.4.6.2 Cream

Bei *Cream* handelt es sich ebenfalls um eine Java-Programmbibliothek. Sie wird von Prof. Naoyuki Tamura an der Kobe University (Japan) entwickelt und als freie Software zur Verfügung gestellt. Im Gegensatz zu Choco sind aber nur wenige Typen von Constraints implementiert. Cream beschränkt sich vor allem auf Constraints die auf Vergleichsrelationen basieren, wie z. B. $=$, $>$, $\geq$, $\neq$. Mit der Implementierung einer Problemformulierung auf Basis von Cream lässt sich zeigen, dass die Formulierung nur wenig Voraussetzungen an Constraint Solver stellt und damit auch von anderen Bibliotheken unterstützt wird. Durch die breite Unterstützung von Programmbibliotheken wird der Austausch des Constraint Solvers vereinfacht und es entsteht keine starke Abhängigkeit zu einer bestimmten Bibliothek. Ein Vorteil von Cream gegenüber Choco ist die Dateigröße. Sie ist bei Cream (52 KB) um ein Vielfaches kleiner als bei Choco (2,5 MB).

6.4.6.3 Eindimensionaler Viewpoint

Der eindimensionale Viewpoint wurde auf Basis von Choco implementiert. Cream fehlt das nvalue-Constraint zur Umsetzung der Bewertungsfunktion. In Choco ist dieses Constraint zwar auch nicht vorhanden, jedoch kann mit Hilfe anderer Constraints dessen Funktion nachgestellt werden (Occurrence- oder Global-Cardinality-Constraint).

Die Implementierung des eindimensionalen Viewpoints beinhaltet das Verfahren der dynamischen Einschränkung der Wertebereich, das dem Symmetry Breaking dient (siehe 6.4.5.3). Zudem wurden die Verfahren zur statischen Einschränkung der Wertebereiche implementiert (siehe 6.4.4).

6.4.6.4 Zweidimensionaler Viewpoint

Der zweidimensionale Viewpoint benötigt nur Constraints basierend auf Vergleichsrelationen und konnte deshalb sowohl mit Choco als auch mit Cream implementiert werden. Da Cream keine Reified Constraints bietet, können mit Cream keine Disjunktionen von Constraints definiert werden. Deshalb wurden die Isolationsvorgaben auf Basis der alternativen Formulierungen, die ohne Disjunktionen auskommen, implementiert.

Die Implementierung des zweidimensionalen Viewpoints mit Cream beinhaltet den 0/1-Viewpoint zur Vermeidung von Wertesymmetrien (siehe 6.4.5.4). Außerdem wurden die beiden Verfahren zur statischen Einschränkung der Wertebereiche implementiert (siehe 6.4.4). Die Implementierung mit Choco enthält, zusätzlich zu den in Cream implementierten Verfahren, das Verfahren zur dynamischen Einschränkung der Wertebereiche (siehe 6.4.5.3).

6.5 Evaluierung

Das Laufzeitverhalten wurde anhand von Implementierungen mit Hilfe der Programmbibliotheken Cream 1.0.6 [Tam] und Choco 2.1.0 [cho] untersucht. Dazu kam ein Computer mit dem Prozessor *Intel Core 2 Duo E4500* (2,2 GHz) und 2 GB Arbeitsspeicher zum Einsatz. Als Betriebssystem wurde Debian GNU/Linux 5.0.3 (amd64) eingesetzt. Es wurde die Laufzeitumgebung des Sun Java Development Kit 6 Update 13 (Standard Edition) für Linux x64 verwendet.

Für die Evaluation wurden mehrere Testfälle erstellt, die unterschiedliche Isolationsvorgaben definieren. Es wurden nur grundlegende Isolationsvorgaben in den Definitionen verwendet, da die erweiterten Vorgaben in Grundlegende übersetzt werden können. Die Isolationsvorgaben wurden als Eingabe für die, in Kapitel 6 vorgestellten, Implementierungen verwendet

und die Laufzeit der Lösungssuche aufgezeichnet. Zudem wurde überprüft, ob die ausgegebenen Lösungen korrekt sind, also eine, für den jeweiligen Testfall, optimale Lösung darstellen.

Die Auflösung der Isolationsvorgaben muss stattfinden bevor Anwendungen gestartet werden können. Deshalb ist es wünschenswert, dass sie möglichst schnell vonstattengeht. Denn zum einen ist die dafür aufgewandte Rechenzeit nicht für die Anwendungen nutzbar. Zum anderen möchte der Benutzer möglicherweise kontrollieren, ob der Start der Anwendungen reibungslos funktioniert. Eine langandauernde Auflösung der Isolationsvorgaben würde eine lästige Wartezeit und eine Unterbrechung des Arbeitsablaufs bedeuten. Die Dauer für den Vorgang sollte sich also in der Größenordnung von Sekunden oder maximal Minuten bewegen, aber keinesfalls eine oder mehrere Stunden betragen.

Anhand der Laufzeitmessungen soll eine Aussage über die maximale Anzahl an Isolationsstufen n und Anwendungen m getroffen werden, die von einer Implementierung *unterstützt* werden. Eine Anzahl wird von einer Implementierung unterstützt, wenn die Lösungssuche für ein Problem, das n Stufen und m Anwendungen umfasst, unabhängig von den Isolationsvorgaben, nach einer zumutbaren Zeit terminiert.

Eine Forderung nach einer bestimmten Mindestanzahl an Anwendungen oder Isolationsstufen, die jede Implementierung unterstützen muss, lässt sich nur schwer begründen, da sie stark vom jeweiligen Anwendungsfall abhängig sind. Im Fall der Anwendungsanzahl kann über die Prozessorleistung und damit über die Prozessorarchitekturen argumentiert werden. Jedoch stellt nicht für jede Anwendung die Taktrate die maßgebende Größe dar. Auch die Geschwindigkeit des Speicherzugriffs kann eine limitierende Größe sein.

Es macht Sinn Anwendungen eigene Prozessorkerne zuzweisen, um Ressourcenkonflikte zu minimieren. Wenn man von der Prozessorleistung als maßgebende Größe ausgeht, unterstützt ein heutiger Desktop-PC mit Multi-Core-Prozessor die nahezu konfliktfreie Ausführung von zwei bis acht Anwendungen [BDM09]. Zukünftige Many-Core-Architekturen versprechen eine weit höhere Parallelisierbarkeit mit Hundert bis Tausend Kernen [Bor07]. Die unterstützte Anzahl an Anwendungen sollte aus Sicht der Prozessorarchitekturen also bei ca. acht Anwendungen liegen. Im Licht der zukünftigen Many-Core-Architekturen erscheinen jedoch auch weit höhere Zahlen als sinnvoll.

In Abschnitt 6.5.1 werden die Testfälle vorgestellt. In den nachfolgenden Abschnitten sind die Ergebnisse der Laufzeitmessungen zusammengefasst.

6.5.1 Testfälle

Folgende Testfälle wurden für die Evaluation verwendet. Sie sind in Abhängigkeit der Anzahl an Anwendungen und Isolationsstufen definiert.

6.5.1.1 Ausnutzen aller Isolationsstufen

Jede Anwendung wird durch eine Isolationsvorgabe auf eine bestimmte Stufe beschränkt, so dass sie gleichmäßig über die Isolationsstufen verteilt werden. Die erste Anwendung auf die ersten Stufe beschränkt, die zweite Anwendung auf die zweite Stufe, usw. Gibt es mehr Anwendungen als Stufen beginnt die Zuordnung wieder von vorn. Bei n Isolationsstufen wird Anwendung i also der Stufe $(i \bmod n)$ zugeordnet. Da die Nachbarschaft zwischen Anwendungen nicht ausgeschlossen wird, werden alle Anwendungen einer Stufe derselben Isolationsumgebung zugeordnet.

Dieser Fall kann als einfach zu lösen angesehen werden, da die Isolationsstufe für jede Anwendung bereits vorgegeben ist. Bei der Lösungssuche muss nur noch festgestellt werden, dass alle Anwendungen einer Stufe in derselben Umgebung unterzubringen sind, um die optimale Lösung zu erreichen.

6.5.1.2 Einsatz aller Isolationsvorgaben

Dieser Testfall setzt alle grundlegene Isolationsvorgaben ein, die im System implementiert sind (siehe 6.1). Dabei handelt es sich um folgende Isolationsvorgaben:

- Anwendung a muss in einer anderen Isolationsumgebung ausgeführt werden als Anwendung b.
- Eine Anwendung muss in einer Isolationsumgebung ausgeführt werden, die einer Isolationsstufe aus einer bestimmten Menge von Isolationsstufen angehört.
- Anwendung a muss mit Anwendung b in einer Isolationsumgebung ausgeführt werden.

In Abhängigkeit der Anzahl an Anwendungen und Isolationsstufen werden die obigen Vorgaben unterschiedlich kombiniert. In Algorithmus 6.2 wird aufgezeigt wie die Isolationsvorgaben für diesen Testfall gebildet werden. Es werden mindestens vier Anwendungen und zwei Isolationsstufen benötigt.

6.5.1.3 Gegenseitiger Ausschluss der Nachbarschaft

Für jede Anwendung a gilt: a ist in einer anderen Isolationsumgebung als alle anderen Anwendungen. Zudem wird a in der stärksten Isolationsstufe ausgeführt. In einer Lösung ist also jede Anwendung in einer eigenen Isolationsumgebung der stärksten Stufe. Dieser Testfall weist eine hohe Wertesymmetrie auf (siehe). Es entstehen viele Lösungen, die sich nur darin unterscheiden, dass die Umgebungs-IDs der Anwendungen vertauscht sind.

Algorithmus 6.2 Einsatz aller Isolationsvorgaben

m: Anzahl Anwendungen
$A = [1; m]$: Menge der Anwendungen
n: Anzahl Isolationsstufen
$S = [1; n]$: Menge der Isolationsstufen
I: Menge der Isolationsvorgaben
// Hilfsvariable für Isolationsstufen
$s_1 \leftarrow 1$
for $i = 1$ to $(m - 3)$ step 4 **do**
  // Hilfsvariablen für Anwendungen
  $a \leftarrow i$; $b \leftarrow i + 1$; $c \leftarrow i + 2$; $d \leftarrow i + 3$
  $I \leftarrow I \cup \{(a$ ist in einer anderen Umgebung als $d)\}$
  $I \leftarrow I \cup \{(b$ ist in der gleichen Umgebung wie $c)\}$
  $I \leftarrow I \cup \{(d$ ist in einer anderen Umgebung wie $b)\}$
  $I \leftarrow I \cup \{(d$ ist in einer anderen Umgebung wie $c)\}$
  // Falls es die Stufe mit der ID $(s_1 + 1)$ nicht gibt, wird wieder bei Stufe 1 begonnen
  if $s_1 + 1 > n$ **then**
    $s_1 \leftarrow 1$
  end if
  $s_2 \leftarrow s_1 + 1$
  // Falls es die Stufe $(s_1 + 2)$ nicht gibt, wird s_3 die gleiche Stufe zugewiesen wie s_2
  if $s_1 + 2 \leq n$ **then**
    $s_3 \leftarrow s_1 + 2$
  else
    $s_3 \leftarrow s_2$
  end if
  $I \leftarrow I \cup \{(a$ ist in Stufe s_1 $)\}$
  $I \leftarrow I \cup \{(b$ ist in Stufe s_1, s_2 oder s_3 $)\}$
  $I \leftarrow I \cup \{(c$ ist in Stufe s_2 oder s_3 $)\}$
  $I \leftarrow I \cup \{(d$ ist in Stufe s_2 oder s_3 $)\}$
  $s_1 \leftarrow s_1 + 3$
end for

Eine Variante dieses Testfalls kann durch Weglassen der Beschränkung auf die stärkste Isolationsstufe erzeugt werden. In einer optimalen Lösung ist eine Anwendung in der Isolationsumgebung der ersten Stufe *Keine Isolation* untergebracht und der Rest der Anwendungen befindet sich in Stufe 2, jeweils in einer eigenen Umgebung.

6.5.1.4 Widersprüchliche Isolationsvorgaben

In diesem Testfall werden Isolationsvorgaben definiert, die sich widersprechen, also nicht erfüllbar sind. Das lässt sich mit nur zwei Isolationsvorgaben bewerkstelligen. Einerseits wird die Vorgabe gestellt, dass Anwendung *a* in der selben Isolationsumgebung ausgeführt wird, wie Anwendung *b*. Andererseits soll *a* in der selben Isolationsumgebung ausgeführt werden, wie *b*.

Die Lösungssuche sollte bei unerfüllbaren Vorgaben möglichst schnell abbrechen, da dies die Grundlage für das Optimierungsverfahren *Iterative Vergrößerung der Wertebereiche* ist (siehe 6.4.4.2).

6.5.2 Laufzeitmessungen ohne Optimierung der Lösungssuche

Die im Folgenden vorgestellten Laufzeitmessungen wurden mit einer konstanten Anzahl von fünf Isolationsstufen und einer steigenden Anzahl an Anwendungen durchgeführt. Dabei wurden weder Verfahren zur Beschränkung der Wertebereiche noch Symmetry-Breaking-Verfahren eingesetzt. Die Ergebnisse der Messungen werden dabei für jeden Testfall und jede Implementierung aufgeführt. Die Datenpunkte zeigen die Mittelwerte aus zehn Messungen, wenn nicht anders angegeben. Die Fehlerbalken zeigen das 95%-Konfidenzintervall der dargestellten Mittelwerte. Da die Implementierungen in fast allen Fällen gute Laufzeiten bei drei oder weniger Anwendungen zeigen, werden hier nur die Messungen ab vier Anwendungen vorgestellt.

6.5.2.1 Ausnutzen aller Isolationsstufen

Die Messergebnisse für diesen Testfall sind in Abbildung 6.2 dargestellt. Die Choco-Implementierung des 2D-Viewpoints schneidet am schlechtesten ab. Bis sechs Anwendungen zeigt sie noch eine akzeptable Laufzeit von unter zehn Sekunden. Die Messung mit sieben Anwendungen wurde nach über fünf Stunden abgebrochen.

Die Cream-Implementierung des 1D-Viewpoints zeigt bis 18 Anwendungen sehr gute Laufzeiten von unter einer Zehntelsekunde. Ab 19 Anwendungen überschreitet das Maximum der Bewertungsfunktion des 2D-Viewpoints den höchsten, von Cream unterstützten, Wert für ganze Zahlen (Integer). Deshalb können nur Messungen bis maximal 18 Anwendungen durchgeführt werden.

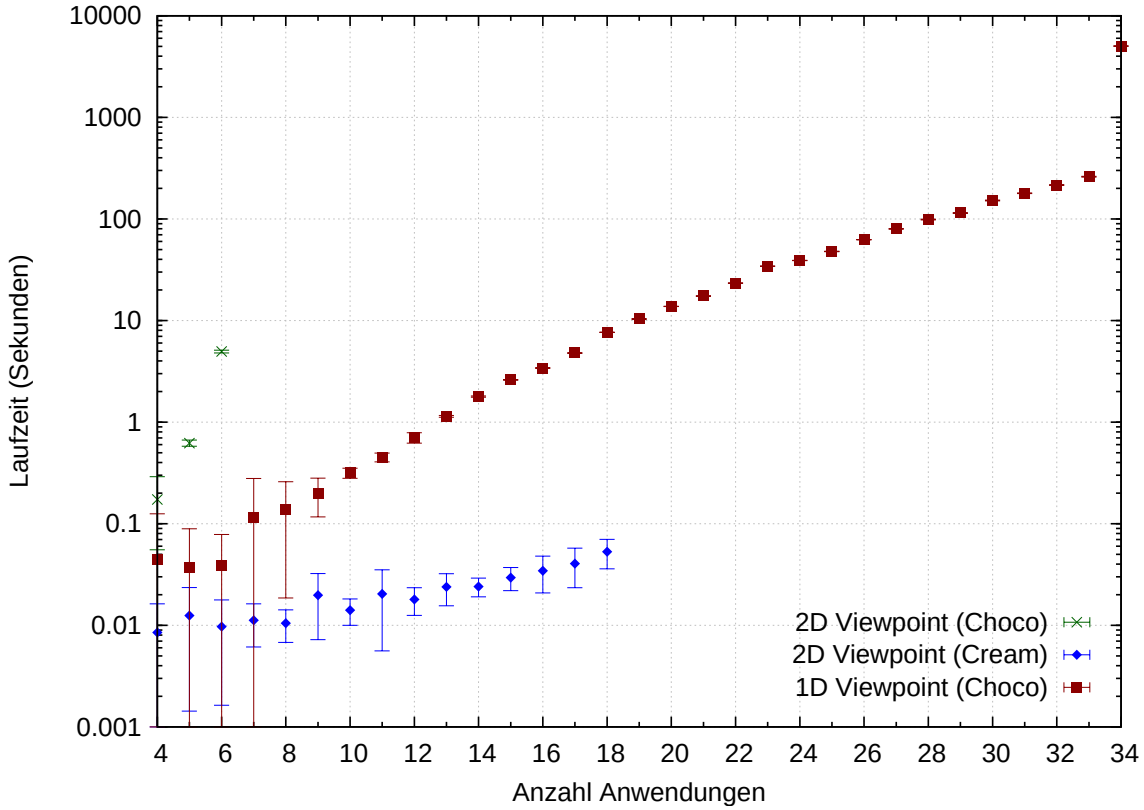

Abbildung 6.2: Ausnutzen aller Isolationsstufen (ohne Optimierung)

Die Implementierung des 1D-Viewpoints zeigt eine etwas schlechtere Laufzeit als die Cream-Implementierung. Jedoch bewegt sie sich bis 18 Anwendungen auch noch unterhalb der 10-Sekunden-Marke. Bei 33 Anwendungen benötigt der 1D-Viewpoint bereits über vier Minuten für die Lösungssuche. Die Messung für 34 Anwendungen wurde nur einmal durchgeführt, da die Laufzeit 1 Std. 24 Min. beträgt.

6.5.2.2 Einsatz aller Isolationsvorgaben

In Abbildung 6.3 sind die Messergebnisse für diesen Testfall dargestellt.

Der Algorithmus des Testfalls legt die Isolationsvorgaben iterativ für jeweils vier Anwendungen gleichzeitig fest (siehe 6.5.1.2). Falls die Anzahl der Anwendungen m nicht restlos durch vier teilbar ist, bleiben $(m \bmod 4)$ Anwendungen übrig, für die keine Isolationsvorgaben definiert werden.

Die Cream-Implementierung des 2D-Viewpoint kommt mit diesen unbeschränkten Anwendungen nicht gut zurecht. Das ist daran zu erkennen, dass die Laufzeiten stark ansteigen

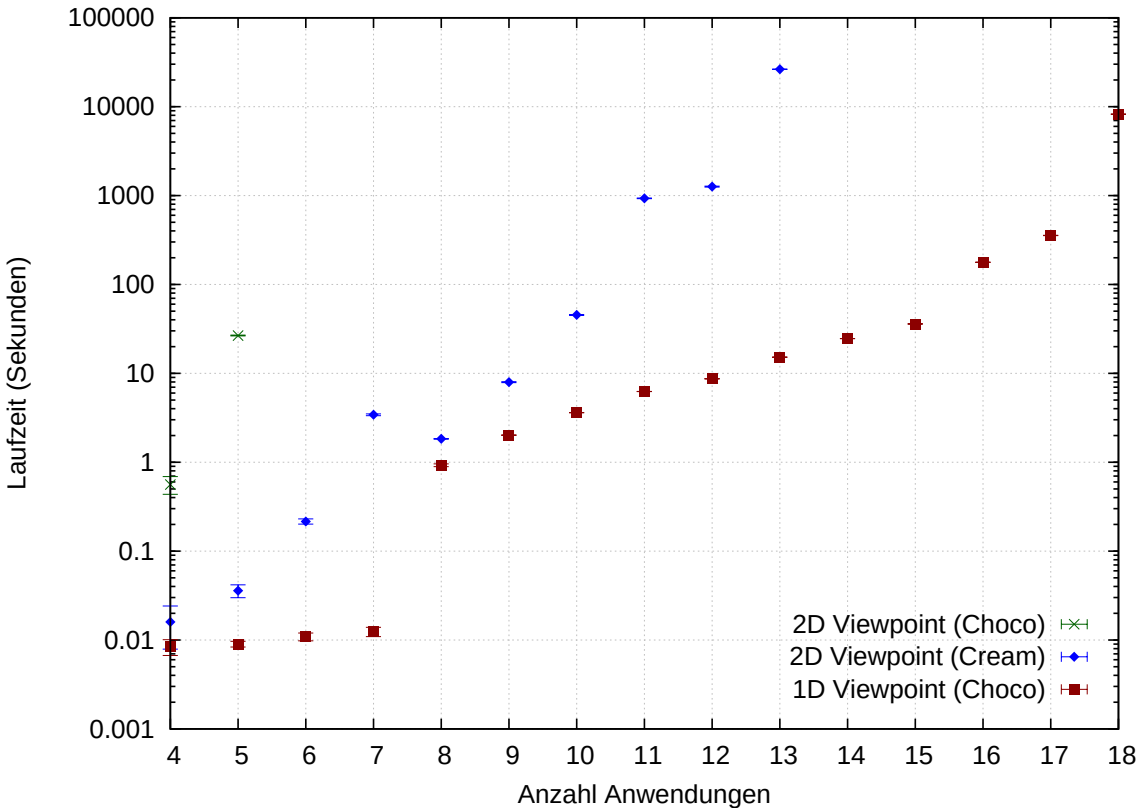

Abbildung 6.3: Einsatz aller Isolationsvorgaben (ohne Optimierung)

je größer die Anzahl der unbeschränkten Anwendungen wird. Sobald die Gesamtanzahl an Anwendungen wieder einen durch vier teilbaren Wert erreicht, ist eine Abnahme der Steigung zu erkennen, wodurch ein Zick-Zack-Muster entsteht.

Bei der Implementierung des 1D-Viewpoints ist ein solches Verhalten nicht zu erkennen. Hier liegt der Fall umgekehrt: Sobald die Gesamtanzahl einen durch vier teilbaren Wert erreicht, nimmt die Steigung zu. Dies ist bei 8 und 16 Anwendungen zu erkennen, jedoch nicht bei 12 Anwendungen. Deshalb kann hier keine klare Aussage getroffen werden, ob das Verhalten tatsächlich auf die Anzahl der Anwendungen ohne Isolationsvorgaben zurückzuführen ist.

Die Laufzeit der Cream-Implementierung des 2D-Viewpoints bewegt sich bis neun Anwendungen noch unterhalb von zehn Sekunden. Bei 13 Anwendungen beträgt die Laufzeit bereits 7 Std. 20 Min. Diese Messung wurde nur einmal durchgeführt.

Die Choco-Implementierung zeigt bereits bei fünf Anwendungen eine Laufzeit von über zehn Sekunden. Die Lösungssuche mit sechs Anwendungen wurde nach über acht Stunden abgebrochen.

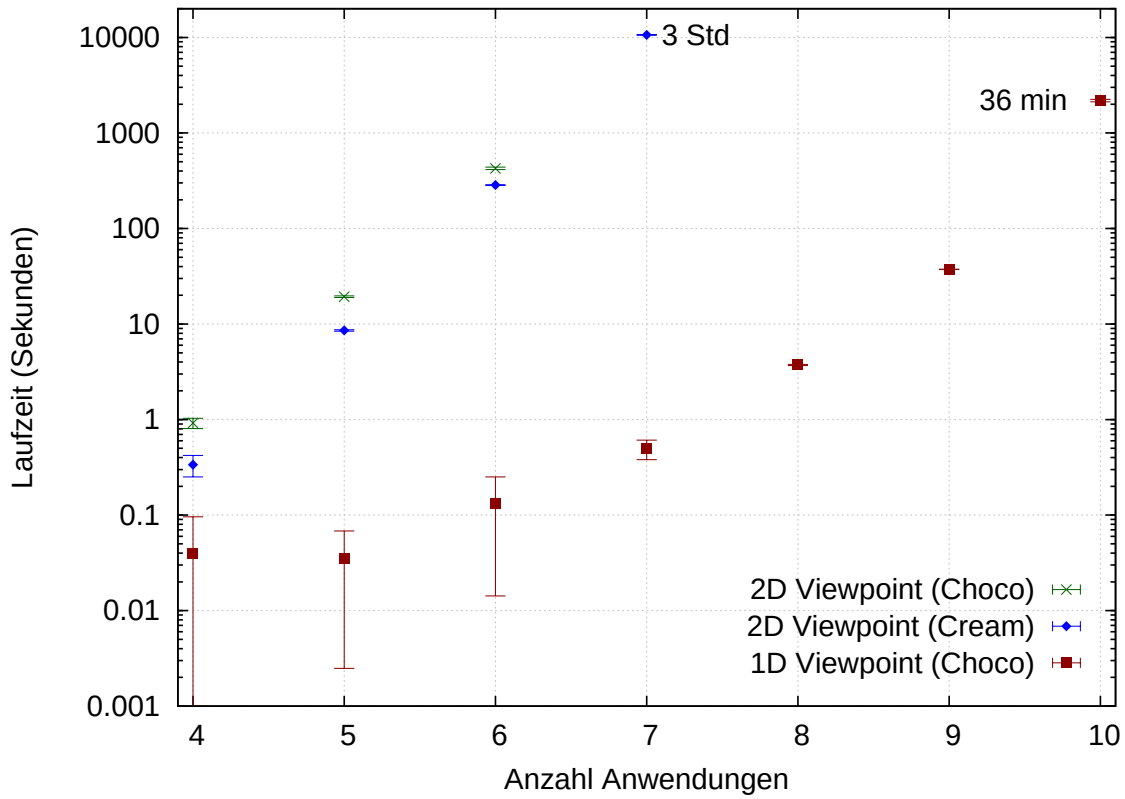

Abbildung 6.4: Gegenseitiger Ausschluss (ohne Optimierung)

Die Implementierung des 1D-Viewpoints überschreitet erst bei 13 Anwendungen die 10-Sekunden-Marke. Bei 18-Anwendungen beträgt die Laufzeit 2 Std. und 17 Min, weshalb die Messung nur einmal durchgeführt wurde.

6.5.2.3 Gegenseitiger Ausschluss der Nachbarschaft

Abbildung 6.4 zeigt die Messergebnisse für diesen Testfall. Es wurden zehn Messungen pro Datenpunkt durchgeführt, ausgenommen ist der Punkt mit sieben Anwendungen der Cream-Implementierung des 2D-Viewpoints und der Punkt mit zehn Anwendungen der Implementierung des 1D-Viewpoints. Für Ersteren wurde nur eine Messung durchgeführt und für Letzteren wurden drei Messungen durchgeführt.

Bei diesem Testfall machen sich die Auswirkungen der Wertesymmetrie stark bemerkbar. Mit zunehmender Anzahl an Anwendungen steigt auch die Anzahl an äquivalenten Lösungen, die durch die Wertesymmetrie erzeugt werden (siehe 6.4.5.1). Es zeigt sich, dass die Implementierung des 1D-Viewpoints effizienter läuft. Bis neun Anwendungen bleibt sie

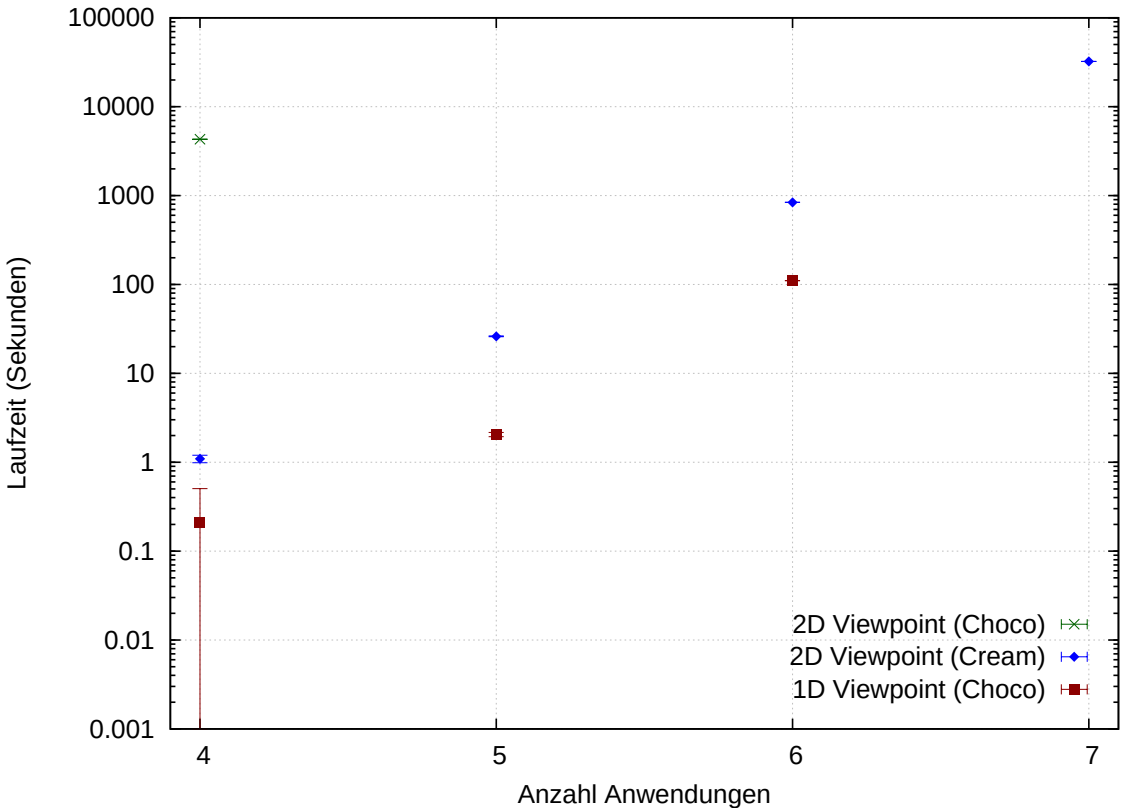

Abbildung 6.5: Gegenseitiger Ausschluss ohne Stufenvorgaben (ohne Optimierung)

noch unter einer Minute. Wohingegen die Implementierungen des 2D-Viewpoints schon bei sechs Anwendungen mehrere Minuten benötigen.

Keine Vorgaben zur Isolationsstufe

Die Laufzeiten für die Variante ohne Beschränkung der Anwendungen auf eine bestimmte Isolationsstufe sind in Abbildung 6.5 dargestellt. Die Verschlechterung der Laufzeiten, im Vergleich zur anderen Variante, lässt sich durch den vergrößerten Suchraum erklären. Da die Anwendungen nicht mehr auf die stärkste Stufe beschränkt werden, sind nun auch Lösungen im Suchraum enthalten, die die Anwendungen auf anderen Stufen oder auf unterschiedlichen Stufen unterbringen.

Die Messung der Choco-Implementierung des 2D-Viewpoints wurde nur einmal durchgeführt. Die Messung mit sieben Anwendungen der Cream-Implementierung wurde ebenfalls nur einmal durchgeführt. Die Laufzeitmessung für den 1D-Viewpoint wurde bei sieben Anwendungen nach 14 Stunden abgebrochen.

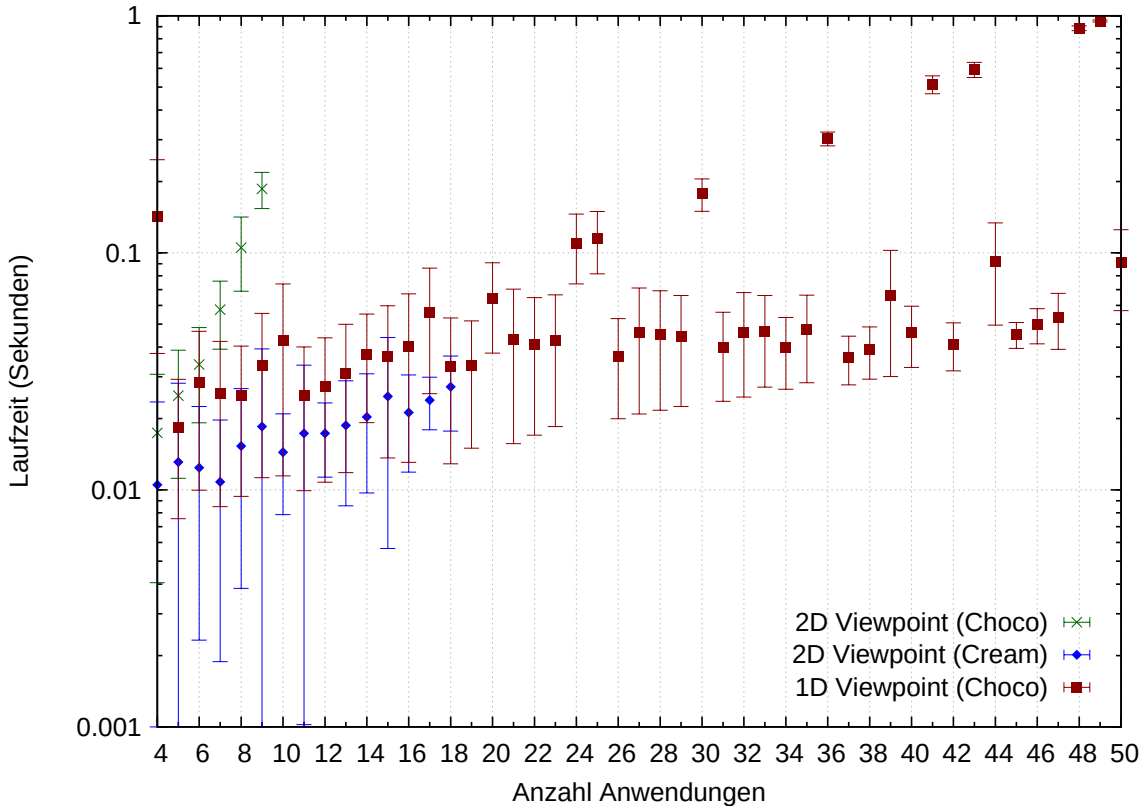

Abbildung 6.6: Widersprüchliche Isolationsvorgaben (ohne Optimierung)

6.5.2.4 Widersprüchliche Isolationsvorgaben

Abbildung 6.6 zeigt die Ergebnisse der Laufzeitmessungen für diesen Testfall. Die Cream-Implementierung und die Implementierung des 1D-Viewpoint zeigen sehr gute Resultate. Cream beendet die Lösungssuche bis zum Maximum von 18 Anwendungen unter einer Zehntelsekunde. Bei 19 Anwendungen überschreitet das Maximum der Bewertungsfunktion des 2D-Viewpoints den höchsten, von Cream unterstützten, Wert für ganze Zahlen (Integer). Der 1D-Viewpoint durchbricht selbst bei 50 Anwendungen nicht die 1-Sekunden-Marke. Die Choco-Implementierung des 2D-Viewpoints zeigt bis neun Anwendungen ebenfalls sehr gute Laufzeiten. Die Messung für zehn Anwendungen wurde jedoch nach 4 Std. 15 Min. ohne Ergebnis abgebrochen.

6.5.2.5 Fazit der Laufzeitmessungen ohne Optimierung der Lösungssuche

Bei den Laufzeitmessungen hat sich die Implementierung des 1D-Viewpoint als Favorit herauskristallisiert. Zwar bietet die Cream-Implementierung des 2D-Viewpoint teilweise

kürzere Laufzeiten, z. B. im Testfall *Ausnutzen aller Isolationsstufen* (siehe 6.5.2.1), jedoch zeigt der 1D-Viewpoint in diesen Fällen ebenfalls gute Laufzeiten. Außerdem unterstützt der 1D-Viewpoint eine höhere Anzahl an Anwendungen. Die Cream-Implementierung ist auf maximal 18 Anwendungen begrenzt, da bei einer größeren Anzahl das Maximum der Bewertungsfunktion den höchsten, von Cream unterstützten, Variablenwert übersteigt.

Der Testfall *Gegenseitiger Ausschluss der Nachbarschaft* hat gezeigt, dass die Lösungssuche ohne Optimierung auch bei einer geringen Anzahl an Anwendungen lange Laufzeiten verursachen kann. Selbst die 1D-Implementierung bewältigt nur max. acht Anwendungen in zehn Sekunden. In der Variante ohne Vorgabe der Isolationsstufe sind es sogar nur fünf Anwendungen. Dieser Testfall stellt damit auch den Worst Case dar.

Auf Basis der Laufzeitmessungen des Worst Case kann eine Aussage über die maximale Anzahl an Anwendungen getroffen werden, die von den Implementierungen unterstützt werden, wenn fünf (oder weniger) Isolationsstufen zur Verfügung stehen. Die 1D-Implementierung unterstützt maximal fünf bis sechs Anwendungen, wobei sechs Anwendungen bereits eine Laufzeit von fast zwei Minuten bedeuten. Diese Zeitspanne wird von einem wartenden Benutzer möglicherweise als zu lang empfunden. Die Cream-Implementierung des 2D-Viewpoint unterstützt maximal vier bis fünf Anwendungen. Bei sechs Anwendungen benötigt sie bereits fast eine Viertelstunde.

6.5.3 Laufzeitmessungen mit Optimierung der Lösungssuche

Im vorigen Abschnitt wurde der Testfall *Gegenseitiger Ausschluss* als Worst Case identifiziert. Die im Folgenden vorgestellten Laufzeitmessungen beschränken sich deshalb auf diesen Testfall. Die Implementierung des 1D-Viewpoint zeigte insgesamt die besten Ergebnisse, weshalb sich die weitere Evaluierung auf deren Optimierung konzentriert.

6.5.3.1 Statische Einschränkung der Wertebereiche

Durch die Einschränkung der Wertebereiche kann der Suchraum eines Problems verkleinert werden, was zu einer verbesserten Laufzeit führen kann. Abbildung 6.7 zeigt die Laufzeitmessungen der Implementierung des 1D-Viewpoint für den Testfall *Gegenseitiger Ausschluss der Nachbarschaft*. Dabei wird die Kombination der Verfahren zur statischen Einschränkung der Wertebereiche eingesetzt (siehe 6.4.4.3). Die Laufzeit verbessert sich dadurch sichtbar. Die Dauer der Lösungssuche für zehn Anwendungen sinkt von 36 min auf 4 min.

Auch die Laufzeiten für die Variante des Testfalls, der keine Vorgabe zu Isolationsstufe gibt, zeigen Verbesserungen. Die 10-Sekunden-Marke wird nun erst bei neun Anwendungen überschritten und nicht schon bei sechs Anwendungen.

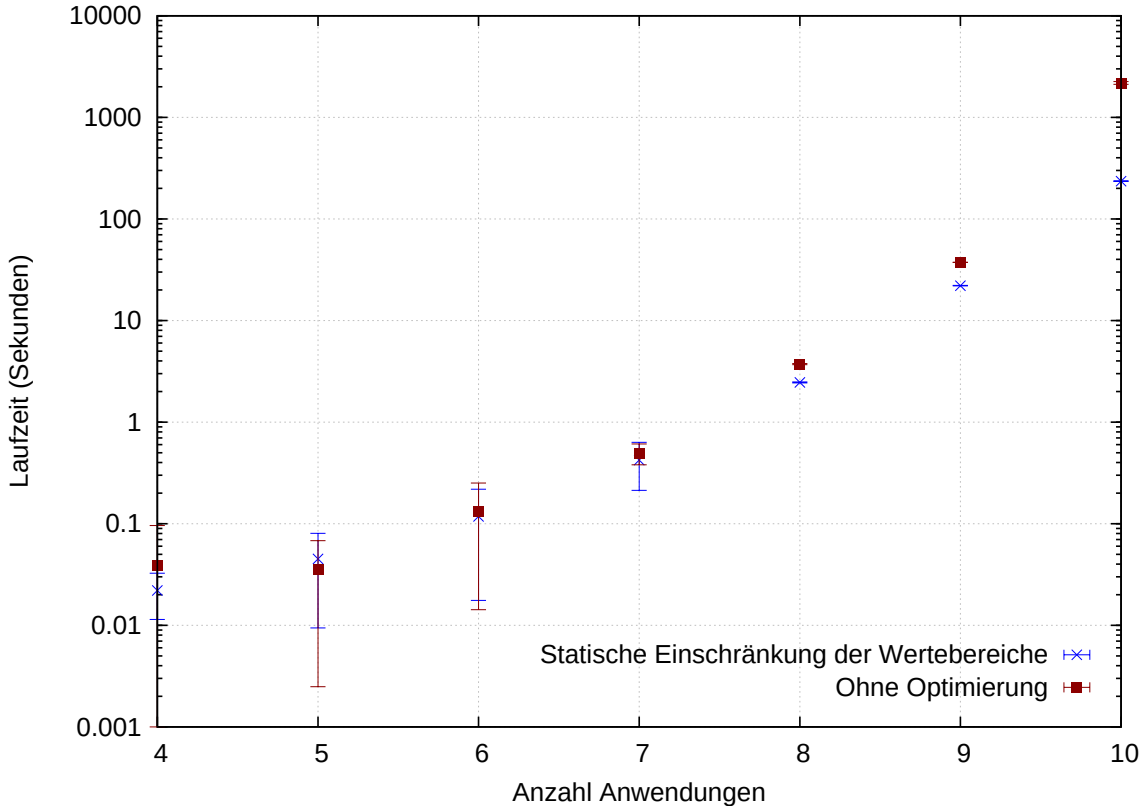

Abbildung 6.7: Gegenseitiger Ausschluss (Statische Einschränkung der Wertebereiche)

6.5.3.2 Symmetry Breaking

Die Implementierung des 1D-Viewpoints beinhaltet das Symmetry-Breaking-Verfahren zur dynamischen Einschränkung der Wertebereiche der Variablen (siehe 6.4.5.3). Die Laufzeiten der Implementierung bei aktiviertem Symmetry Breaking werden in Abbildung 6.9 dargestellt. Dabei handelt es sich wieder um den Testfall *Gegenseitiger Ausschluss der Nachbarschaft*. Das Verfahren bewirkt eine starke Verbesserung der Laufzeiten. Selbst bei 50 Anwendungen bleibt die Laufzeit unter einer Sekunde.

Die Ergebnisse der Variante ohne Beschränkung auf die stärkste Isolationsstufe sind in Abbildung 6.10 dargestellt. Wie bei der statischen Einschränkung der Wertebereich dauert die Lösungssuche erst mit neun Anwendungen länger als zehn Sekunden. Ohne Optimierung der Lösungssuche war dies schon bei sechs Anwendungen der Fall. Insgesamt sind die Ergebnisse jedoch etwas schlechter als bei der statischen Einschränkung der Wertebereiche.

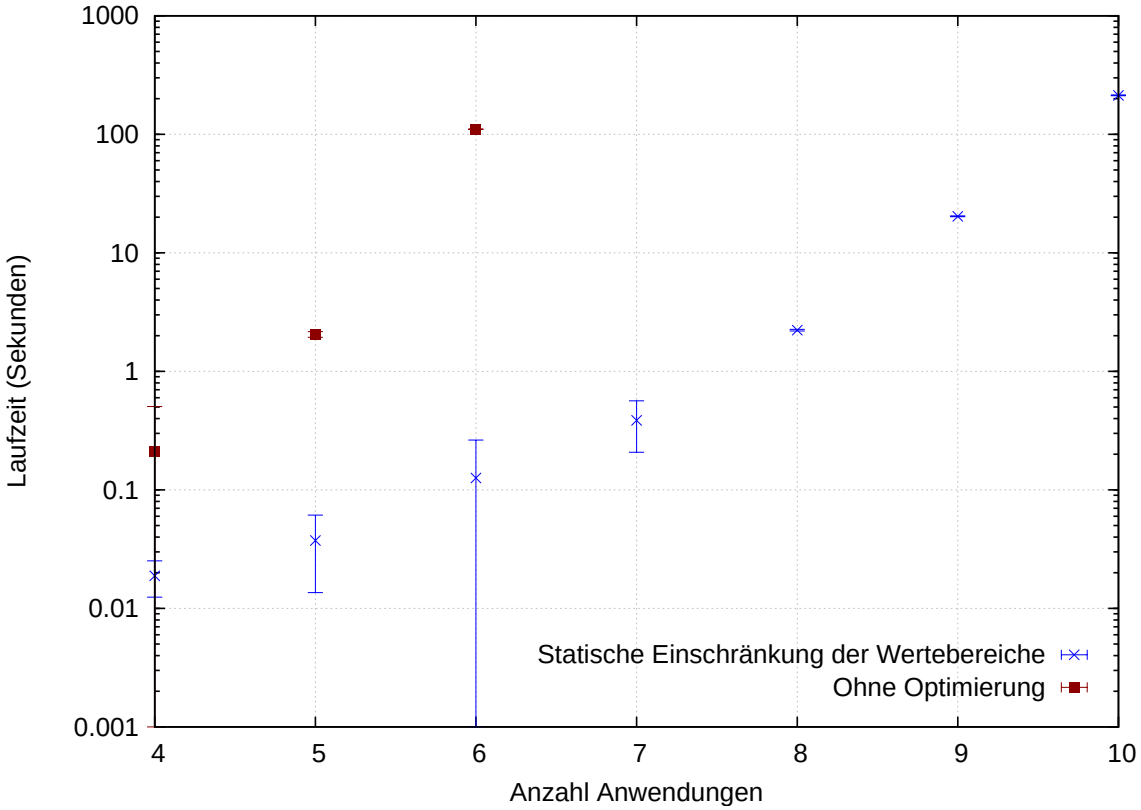

Abbildung 6.8: Gegenseitiger Ausschluss ohne Stufenvorgaben (Statische Einschränkung der Wertebereiche)

6.5.3.3 Kombination von Symmetry Breaking und statischer Einschränkung der Wertebereiche

Die vorgestellten Verfahren zur Optimierung der Lösungssuche haben bereits deutliche Verkürzungen der Laufzeiten erbracht. Durch die Kombination der Verfahren kann eine weitere Verbesserung erzielt werden, wie Abbildung 6.11 zeigt. Als Testfall dient wieder der gegenseitige Ausschluss der Nachbarschaft. Die Laufzeiten verschlechtern sich leicht gegenüber dem getrennten Einsatz des Symmetry Breaking (siehe 6.5.3.2). Die Verschlechterung ist durch den Einfluss des Verfahrens zur statischen Einschränkung der Wertebereiche zu erklären. Die Wertebereiche werden zunächst so eingeschränkt, dass nur eine Isolationsumgebung für die höchste Stufe vorgehalten wird. Da dies für die Auflösung der Isolationsvorgaben nicht ausreicht, wird schrittweise ein Umgebung hinzugefügt bis das nötige Maximum m an Umgebungen erreicht ist, wobei m der Anzahl an Anwendungen entspricht. Diese Vergrößerung des Wertebereichs wird beim getrennten Einsatz des Symmetry Breaking nicht

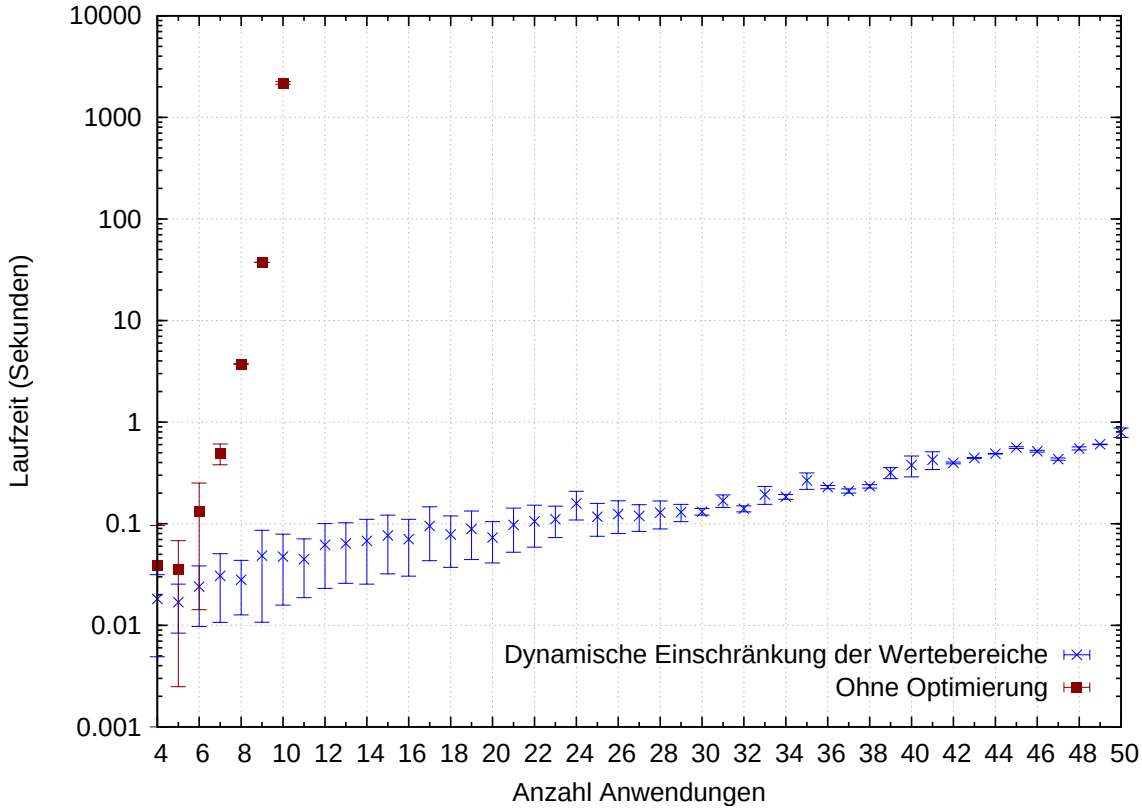

Abbildung 6.9: Gegenseitiger Ausschluss (Dynamische Einschränkung der Wertebereiche)

durchgeführt. Es wird direkt mit dem Maximum an Umgebungen gestartet, wodurch die Lösungssuche in diesem Fall abgekürzt wird.

Die Variante des Testfalls, die keine Vorgaben zu Isolationsstufen macht, zeigt den positiven Einfluss der statischen Einschränkung der Wertebereiche. Bis 36 Anwendungen wird die Dauer von zehn Sekunden bei der Lösungssuche nicht überschritten. Beim getrennten Einsatz der Verfahren wurde die 10-Sekunden-Marke schon bei neun Anwendungen überschritten. Da keine Vorgaben zu den Isolationsstufen der Anwendungen gemacht werden, erweitert das Verfahren zur statischen Einschränkung die Wertebereich um Isolationsumgebungen beginnend bei der schwächsten Stufe. Sobald die schwächste Stufe $(m-1)$ Umgebungen enthält, ist das Problem lösbar. Die potentiellen Wertesymmetrien werden durch das Symmetry-Breaking-Verfahren vermieden. Dadurch kann bei der Variante des Testfalls ohne Stufenvorgaben eine starke Verbesserung erzielt werden.

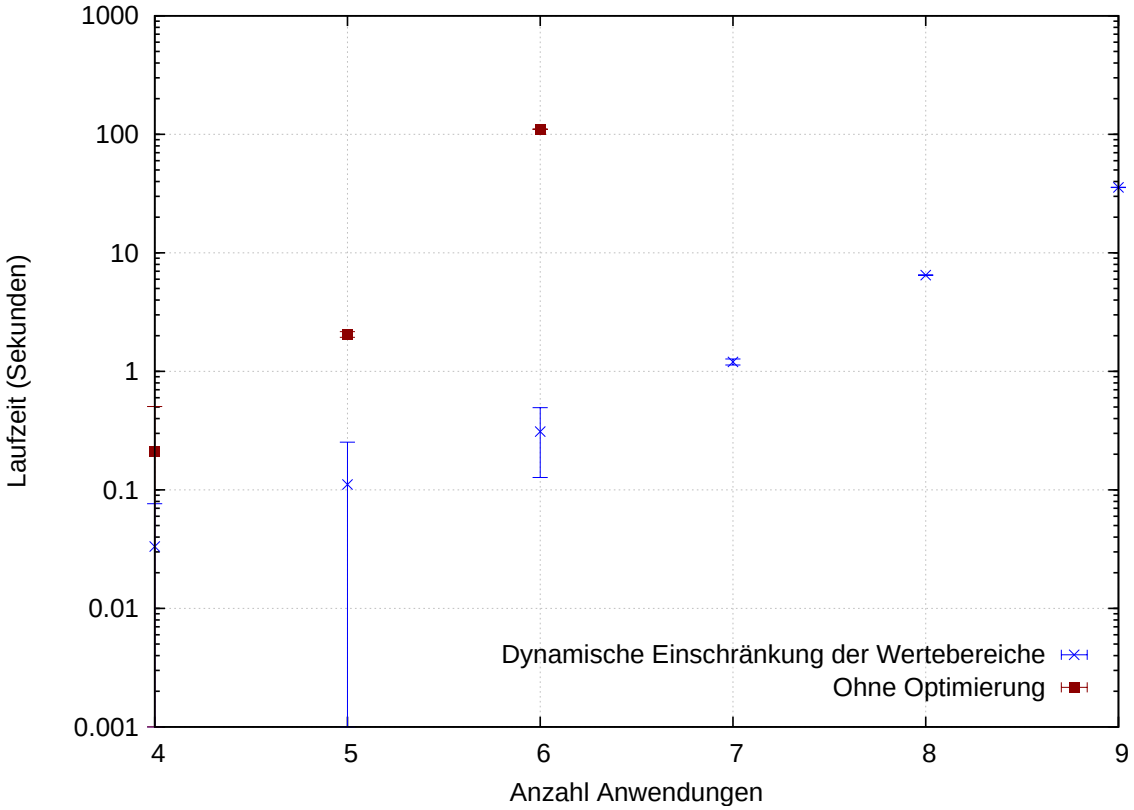

Abbildung 6.10: Gegenseitiger Ausschluss ohne Stufenvorgaben (Dynamische Einschränkung der Wertebereiche)

6.5.4 Fazit

In den vorangegangenen Abschnitten wurden die Implementierungen zur Auflösung der Isolationsvorgaben auf Basis von Constraint Optimization Problems auf ihr Laufzeitverhalten hin untersucht. Es hat sich gezeigt, dass die Implementierung des 1D-Viewpoint für max. 36 Anwendungen und 5 Isolationsstufen, auch im schlimmsten Fall, innerhalb von zehn Sekunden eine Lösung liefert. Dazu ist jedoch die Kombination verschiedener Verfahren zur Optimierung der Lösungssuche nötig.

Es sollte erwähnt werden, dass die Programmbibliothek Choco 2.1.0 bei wiederholter Durchführung der Lösungssuche mit einer großen Anzahl an Anwendungen (> 30) einen hohen Speicherverbrauch (> 500 MB) aufweist. Dies hat bei der Messung des Testfalls *Gegenseitiger Ausschluss der Nachbarschaft* teilweise zu `OutOfMemoryExceptions` geführt. Das Problem kann

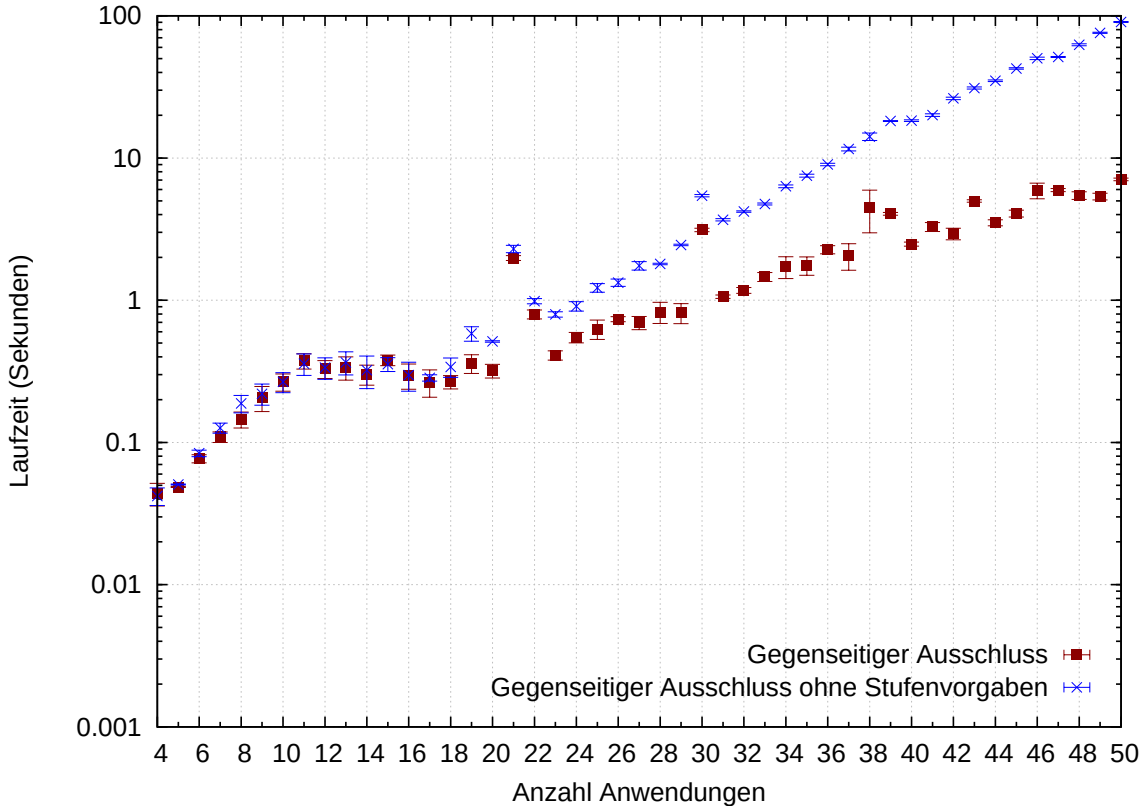

Abbildung 6.11: Kombination von dynamischer und statischer Einschränkung der Wertebereiche

durch einen Neustart der JVM zwischen den Messungen umgangen werden. Die Entwicklern von Choco arbeiten an dem Problem[1].

6.6 Benutzungsschnittstelle zur Definition von Isolationsvorgaben

Im Systemmodell wurden Rollen definiert, die typischerweise unterschiedliche Vorkenntnisse zu Desktop Grids und den Anwendungen haben (siehe 4.1). Zudem unterscheidet sich deren Sichtweise auf die Anwendungen und das Desktop Grid. Folglich werden unterschiedliche Benutzungsschnittstellen für die Definition von Isolationsvorgaben benötigt.

[1]Choco Bug #2947600: http://sourceforge.net/tracker/?func=detail&aid=2947600&group_id=96738&atid=615736

6.6.1 Nutznießer

Der Nutznießer möchte eine Anwendung in einem Desktop Grid ausführen. Sein Interesse gilt vor allem dem Schutz seiner Anwendung. Zudem soll die Anwendung keine Sicherheitsgefahr darstellen, da dies zur Schädigung des Vertrauensverhältnisses gegenüber dem Betreiber des Grids oder den Benutzern führen kann. In beiden Fällen beziehen sich die Isolationsvorgaben auf die Anwendung des Nutznießers. Es bietet sich also an, die Vorgaben zusammen mit der Anwendung zu verteilen. Das kann z. B. durch eine mitgelieferte Datei geschehen, die die Vorgaben enthält. Auf den Wirtsystemen wird die Datei vom Isolationssystem eingelesen und die enthaltenen Vorgaben werden berücksichtigt.

Vom Nutznießer kann angenommen werden, dass er technische Kenntnisse über Software-Entwicklung besitzt oder darauf zurückgreifen kann, da er auch der Urheber seiner Anwendung ist oder Diesen beauftragt hat. Es kann also davon ausgegangen werden, dass die Fähigkeit zur manuellen Erstellung der Konfigurationsdatei mit den Isolationsvorgaben vorhanden ist. Dem Nutznießer muss keine graphische Benutzungsoberfläche zur Verfügung stehen. Die Dokumentation zum Aufbau der Datei sollte ausreichen. Es bietet sich ein menschenlesbares Format, z. B. XML an, da zur Erstellung ein einfacher Texteditor ausreicht.

6.6.2 Benutzer

Der Benutzer stellt seinen Computer als Wirtsystem zur Verfügung, möchte ihn aber auch ungestört anderweitig nutzen. Sein Interesse gilt also vor allem dem Schutz seines Wirtsystems. Je nach Vertrauen zum Nutznießer einer Anwendung möchte er Vorgaben zur Isolation vergeben. Da die Vorgaben von den installierten Anwendungen abhängen, ist es sinnvoll dem Benutzer bei der Installation von Anwendungen die Möglichkeit zu geben deren Isolationsvorgaben zu definieren. Außerdem sollte der Benutzer die Vorgaben einsehen können, die vom Nutznießer mit den Anwendungen mitgeliefert werden.

Der Benutzer zieht im Gegensatz zum Nutznießer meist keinen direkten Nutzen aus der Ausführung der Anwendungen. Es ist davon auszugehen, dass nur eine geringe Bereitschaft besteht, sich Kenntnisse über das Isolationssystem anzueignen. Deshalb sollte die Definition von Isolationsvorgaben möglichst einfach vonstatten gehen. Das kann durch eine graphische Benutzungsoberfläche bewerkstelligt werden. Bei der Installation wird dem Benutzer ein Dialog mit Informationen über die Anwendung angezeigt. Aufgrund der Information kann der Benutzer eine Entscheidung über die gewünschten Isolationsvorgaben treffen. Damit der Benutzer davon nicht überfordert wird, kann eine vereinfachte Oberfläche für die Festlegung der Isolationsvorgaben zur Verfügung gestellt werden. Diese erlaubt dem Benutzer die Festlegung der Isolationsstufe für eine Anwendung z. B. anhand eines Schiebereglers. Der Regler stellt die Isolationsstufen als Ordinalskala dar. Die Auswahl einer bestimmten Stufe wird als Vorgabe interpretiert, die Anwendung in einer eigenen Umgebung dieser Stufe

zu isolieren. Zur weiteren Vereinfachung kann der Schieberegler, statt mit den Namen der Stufen, mit Beschreibungen der Isolationsstärke beschriftet werden, die von *nicht isoliert* bis *stark isoliert* reichen. Wird vom Benutzer eine schwächere Isolationsstufe gewählt als eine vom Nutznießer vorgegebene Stufe, sollte eine Warnung erscheinen, die darauf hinweist.

Fortgeschrittenen Benutzern kann über eine separate Oberfläche der volle Umfang an Isolationsvorgaben zur Verfügung gestellt werden.

6.6.3 Betreiber

Der Betreiber verwaltetet ein Desktop Grid. Hat jedoch an den Anwendungen oder der Infrastruktur nur dann ein direktes Interesse, wenn er selbst Nutznießer oder Benutzer ist. Es besteht jedoch unter Umständen eine Vertrauensbeziehung vom Betreiber zu den Nutznießern und zu den Benutzern. Je nach Organisation des Desktop Grids erhält der Betreiber eine Entlohnung von den Nutznießern für die Ausführung ihrer Anwendungen im Grid. Der Benutzer erwartet vom Betreiber, dass ihm keine schädlichen Anwendungen zur Ausführung auf seinem Wirtsystem zur Verfügung gestellt werden. Eine Verletzung dieser Vertrauensbeziehungen kann also den Verlust von Entlohnung oder von Grid-Ressourcen für den Betreiber bedeuten. Unter Umständen hat der Betreiber also ein indirektes Interesse an dem Schutz aller Anwendungen und Wirtsysteme des Desktop Grids. Dadurch entsteht das Bedürfnis des Betreibers ebenfalls Isolationsvorgaben zu definieren. Dies kann durch das Verteilen einer Datei mit Isolationsvorgaben auf alle Wirtsysteme geschehen. Dadurch kann der Betreiber, je nach Vertrauen zu den Nutznießern, z. B. Vorgaben für eine Mindestisolationsstärke geben. Diese Vorgaben werden, ähnlich wie die Vorgaben des Nutznießers, bei der Installation berücksichtigt. Wählt der Benutzer eine schwächere, als die vom Betreiber vorgeschlagene, Isolationsstufe, wird er darauf hingewiesen.

7 Zusammenfassung und Ausblick

Moderne Desktop-Grid-Plattformen erlauben die Ausführung und Verwaltung mehrerer Anwendungen in einem Desktop Grid. Dadurch entstehen neue Risiken für die Sicherheit und Robustheit der Anwendungen. Durch die Nutzung einer gemeinsamen Plattform wird die Gefahr erhöht, dass sich die Anwendungen gegenseitig beeinflussen. Um Interaktionen zu vermeiden, sollten die Anwendungen isoliert werden.

Neben dem Schutz des Wirtsystems und der Anwendungen wurden in Abschnitt 1.2 weitere Gründe für die Anwendungsisolation identifiziert. Auch die Portabilität der Anwendungen kann von der Isolation profitieren.

Abschnitt 1.3 beschreibt unterschiedliche Anwendungsfälle für die Isolation von Anwendungen.

Der Kommunikationsaufwand mit der Desktop-Grid-Plattform, sowie die zur Verfügung stehende Rechenleistung hängt stark vom Isolationsgrad ab. Zudem existieren Anwendungsfälle, die eine spezielle Art der Isolation erfordern. Deshalb sollte der Isolationsgrad in Abhängigkeit des jeweiligen Anwendungsfalls gewählt werden können, so dass eine Abwägung zwischen Sicherheit und Leistung möglich ist und die Isolationsart auf den Anwendungsfall zugeschnitten werden kann.

In Kapitel 3 werden Arbeiten vorgestellt, die sich ebenfalls mit der Isolation von Desktop-Grid-Anwendungen auseinandersetzen. Die, in den Arbeiten vorgestellten, Systeme sind auf bestimmte Arten der Isolation spezialisiert. Die Abwägung zwischen Sicherheit und Leistung kann nur begrenzt stattfinden. Zudem kann die Isolationsart nicht entsprechend dem jeweiligen Anwendungsfall gewählt werden.

Das in dieser Arbeit vorgestellte System ist für unterschiedliche Isolationsarten konzipiert. Zudem erlaubt es dem Benutzer Vorgaben zur Isolation einer Anwendung zu spezifizieren, so dass der Isolationsgrad je nach Anwendungsfall und Vertrauen zum Urheber der Anwendung wählbar ist. Es wird eine Architektur vorgestellt, die die Verwaltung mehrerer Anwendungen ermöglicht, die auf unterschiedliche Arten isoliert werden können (siehe 4). Trotz Isolation müssen die Anwendungen die Dienste der Desktop-Grid-Plattform nutzen können. Für dieses Problem wird eine Lösung auf Basis von OSGi Remote Services vorgestellt.

In Kapitel 5 werden unterschiedliche Isolationsarten vorgestellt und die Effektivität und Effizienz von zwei Arten untersucht. Es wird ermittelt, welche Schwachstellen durch die

Isolation geschlossen werden können und welche Auswirkungen die Isolation auf den Kommunikationsaufwand und die Rechenleistung hat.

Aus den Vorgaben zur Isolation, die z. B. vom Benutzer des Wirtsystems gestellt werden, muss eine angemessene Isolationsart für jede Anwendung abgeleitet werden. Kapitel 6 beschäftigt sich mit diesem NP-vollständigen Problem. Es werden Ansätze und Implementierungen zur Lösungen des Problems vorgestellt und diese evaluiert.

Ausblick

Das, in dieser Arbeit vorgestellte, Isolationssystem bietet Ansatzpunkte für weitere Untersuchungen. So können weitere Arten der Isolation implementiert und evaluiert werden. Die Evaluierung der Effizienz des Systems kann noch ausgeweitet werden, um z. B. die Auswirkungen der Isolation auf Anwendungen mit intensiver Nutzung des Arbeitsspeichers oder der Datei-Ein- und -Ausgabe zu untersuchen. Um besser auf Angriffe reagieren zu können, die einen hohen Ressourcenverbrauch verursachen, könnte eine System zur Ressourcenüberwachung und -kontrolle in das Isolationssystem integriert werden.

Mit Hinblick auf zukünftige Hardware, die eine hohe Zahl an Prozessorkernen verspricht [Boro7], steht der Ausführung einer hohen Anzahl an Anwendungen, in der Größenordnung von 100, auf einem Wirtsystem nichts mehr im Wege. Um dem Benutzer trotzdem eine einfache Verwaltung der Anwendungen zu ermöglichen, ist eine Gruppierung von Anwendungen oder eine ähnliche Abstraktion sinnvoll. Trotzdem soll der Benutzer weiterhin die Möglichkeit haben, die Isolation an den jeweiligen Anwendungsfall anpassen zu können. Hierbei gilt es zu untersuchen, wie dieser Konflikt am Besten gelöst werden kann. Zum Beispiel ist es in einem solchen Fall sinnvoll eine vereinfachte Implementierung für die Auflösung der Isolationsvorgaben zu verwenden, nicht zuletzt um eine akzeptable Laufzeit zu erreichen. Dazu könnte z. B. die Zurückführung der Auflösung von Isolationsvorgaben auf das Knotenfärbungsproblem herangezogen werden (siehe 6.3).

Es existieren Ansätze die Konzepte des Trusted Computing und des Grid Computing zusammenzubringen Es sollte untersucht werden, wie Trusted Computing in einer Desktop-Grid-Plattform eingesetzt werden kann, die die parallele Ausführung mehrerer Anwendungen ermöglicht, die auf unterschiedliche Arten isoliert werden können. Die Abwehr von Angriffen des Wirtsystems auf die Anwendungen stellt eine attraktive Einsatzmöglichkeit des Trusted Computing dar, um sensiblen Anwendungsdaten zu schützen.

Ein offener Punkt bleibt die Absicherung der Kommunikation zwischen den Anwendungen und der Desktop-Grid-Plattform. Die vorgestellte Lösung zur Kommunikation basiert auf einer Implementierung von OSGi Remote Services, die das TCP-Protokoll einsetzt. Durch einen Port-Scan könnten offene Verbindungen aufgedeckt werden und darauf unberechtigt zugegriffen werden. Es gilt zu untersuchen, wie mit Hilfe von Authentifizierung und

Personal Firewalls eine Abwehr solcher Angriffe bewerkstelligt werden kann. Bisherige Implementierungen der OSGi Remote Services unterstützen entweder gar keine Authentifizierung oder machen Eingriffe in die Anwendungen erforderlich. Dadurch wird jedoch die Transparenz der OSGi Remote Services aufgegeben.

A Beschreibung der Schwachstellen

Im Folgenden werden die Schwachstellen beschrieben, die zur Evaluierung der Effektivität der Isolationsstufen verwendet wurden.

A.1 Liste der Schwachstellen

Eine großer Teil der untersuchten Schwachstellen stammt aus einem Katalog von P. Parrend und S. Frénot [PF07]. Einige Schwachstellen des Katalogs wurden nicht für die Evaluation herangezogen, da sie keine Relevanz für die vorliegende Arbeit besitzen. Dabei handelt es sich zum Beispiel um Schwachstellen, die auf eingebettene Systeme mit stark begrenzten Arbeitsspeicher abzielen. Auch Schwachstellen in Bezug auf Fragment-Bundles wurden nicht untersucht, da Fragment-Bundles über Framework-Grenzen nicht funktionieren, und somit vom Isolationssystem nicht unterstützt werden.

A.1.1 Management Utility Freezing - Infinite Loop

Ein Bundle führt eine Endlosschleife bei der Aktivierung aus.

Angriffsart: Nichtverfügbarkeit, Leistungseinbruch

Auswirkungen *Keine Isolation*: Die OSGi-Management-Konsole ist blockiert. Der Benutzer kann die Konsole nicht mehr zur Verwaltung des Systems verwenden. Zudem wird eine hohe CPU-Last verursacht.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Wenn die Aktivierung des Bundles nach einer gewissen Zeit nicht beendet ist, tritt ein Timeout ein und das System meldet einen Fehler bei der Installation des jeweiligen Bundles. Es wird weiterhin eine hohe CPU-Last verursacht.

Status: *Schwachstelle mittels Überwachung vermeidbar*
Eine Blockierung des Systems kann nicht mehr auftreten. Die hohe CPU-Last kann durch das Zerstören der Isolationsumgebung vermieden werden.

Implementierte Gegenmaßnahme: Die Aktivierung des Bundles wird in einem eigenen Thread angestoßen. Falls die Aktivierung nach einer bestimmten Zeit nicht beendet ist, wird eine Fehlermeldung ausgegeben.

ID: Mb.osgi.4

Quellen: [PF07]

A.1.2 Management Utility Freezing - Thread Hanging

Ein Bundle ruft bei der Aktivierung wiederholt die Methode `Thread.sleep()` auf, wodurch der Aktivierungs-Thread angehalten wird.

Angriffsart: Nichtverfügbarkeit

Auswirkungen *Keine Isolation*: Die OSGi-Management-Konsole ist blockiert. Der Benutzer kann die Konsole nicht mehr zur Verwaltung des Systems verwenden.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Wenn die Aktivierung des Bundles nach einer gewissen Zeit nicht beendet ist, tritt ein Timeout ein und das System meldet einen Fehler bei der Installation des jeweiligen Bundles.

Status: *Schwachstelle geschlossen*

Implementierte Gegenmaßnahme: Die Aktivierung des Bundles wird in einem eigenen Thread angestoßen. Falls die Aktivierung nach einer bestimmten Zeit nicht beendet ist, wird eine Fehlermeldung ausgegeben.

ID: Mb.osgi.5

Quellen: [PF07]

A.1.3 Decompression Bomb

Die komprimierte JAR-Datei eines Bundles enthält Dateien, die unerwartet groß sind im Vergleich zur Größe der JAR-Datei. Beim Entpacken wird dadurch möglicherweise viel Arbeits- und Massenspeicher verbraucht.

Angriffsart: Leistungseinbruch

Auswirkungen *Keine Isolation*: Es entstehen keine Beeinträchtigungen.

Status: *Schwachstelle geschlossen*

Implementierte Gegenmaßnahme: Die JAR-Datei wird vom OSGi-Framework bei der Installation nicht entpackt.

Auswirkungen *Prozess*: siehe Auswirkungen für Stufe *Keine Isolation*

Status: siehe Status von Isolationsstufe *Keine Isolation*

Implementierte Gegenmaßnahme: siehe Gegenmaßnahme von Isolationsstufe *Keine Isolation*

ID: Mb.archive.3

Quellen: [PF07]

A.1.4 Excessive Size of Manifest File

Die Manifest-Datei eines Bundles enthält einen `Import-Package`-Header mit einer große Anzahl an Java-Paketnamen. Bei der Installation kann dadurch eine hohe Verzögerung entstehen, wenn Probleme beim Einlesen der Manifest-Datei entstehen.

Angriffsart: Nichtverfügbarkeit

Auswirkungen *Keine Isolation*: Das OSGi-Framework bricht die Installation des Bundles mit einer Fehlermeldung ab.: `org.osgi.framework.BundleException: The manifest line ‘‘org.osgi.service.obr, org.osgi.service.o’’ is invalid; it has no colon ‘:’ character after the header key.`

Status: *Schwachstelle geschlossen*

Auswirkungen *Prozess*: siehe Auswirkungen für Stufe *Keine Isolation*

Status: siehe Status von Isolationsstufe *Keine Isolation*

ID: Mb.osgi.2

Quellen: [PF07]

A.1.5 Runtime.exec.kill

Ein Bundle nutzt einen Befehl des Betriebssystems (`kill $PPID`), um den Prozess des eigenen OSGi-Frameworks zu beenden.

Angriffsart: Nichtverfügbarkeit

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt Ausführungsrechte (FilePermission `execute`) für die Datei des Befehls (oder für eine Shell des Betriebssystems).

Auswirkungen *Keine Isolation*: Die Isolationsumgebung (das OSGi-Framework) wird zerstört. Alle enthalten Anwendungen werden beendet. Da es sich um das Haupt-Framework handelt, wird auch die Desktop-Grid-Plattform beendet, wodurch auch Anwendungen aus anderen Isolationsumgebungen beeinträchtigt werden.

Status: *Schwachstelle mittels SecurityManager geschlossen*

Indem keine Ausführungsrechte (FilePermission `execute`) an das Bundle vergeben werden, kann die Schwachstelle geschlossen werden. Benötigt die zugehörige Anwendung für die korrekte Funktionsweise Ausführungsrechte für eine große Anzahl an Dateien oder gar beliebige Dateien, gestaltet sich die Umsetzung als schwierig bzw. unmöglich.

Auswirkungen *Prozess*: Die Isolationsumgebung wird zerstört. Alle enthalten Anwendungen werden beendet. Das Wirtsystem oder Anwendungen aus anderen Isolationsumgebungen werden nicht beeinträchtigt.

Status: *Schwachstelle geschlossen*

ID: Mb.native.1

Quellen: [PF07]

A.1.6 System.exit

Ein Bundle ruft die Methode `System.exit(int)` auf und beendet dadurch die Java-Laufzeitumgebung.

Angriffsart: Nichtverfügbarkeit

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt die Berechtigung RuntimePermission `exitVM`.

Auswirkungen *Keine Isolation*: Die Isolationsumgebung (das OSGi-Framework) wird zerstört. Alle enthalten Anwendungen werden beendet. Da es sich um das Haupt-Framework handelt, wird auch die Desktop-Grid-Plattform beendet, wodurch auch Anwendungen aus anderen Isolationsumgebungen beeinträchtigt werden.

Status: *Schwachstelle mittels SecurityManager geschlossen*
Indem die Berechtigung RuntimePermission `exitVM` nicht an das Bundle vergeben wird, kann die Schwachstelle geschlossen werden.

Auswirkungen *Prozess*: Die Isolationsumgebung wird zerstört. Alle enthalten Anwendungen werden beendet. Das Wirtsystem oder Anwendungen aus anderen Isolationsumgebungen werden nicht beeinträchtigt.

Status: *Schwachstelle geschlossen*

ID: Mb.java.1

Quellen: [PF07]

A.1.7 Runtime.halt

Ein Bundle ruft die Methode `Runtime.halt(int)` auf und beendet dadurch die Java-Laufzeitumgebung.

Angriffsart: Nichtverfügbarkeit

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt die Berechtigung RuntimePermission `exitVM`.

Auswirkungen *Keine Isolation*: Die Isolationsumgebung (das OSGi-Framework) wird zerstört. Alle enthalten Anwendungen werden beendet. Da es sich um das Haupt-Framework handelt, wird auch die Desktop-Grid-Plattform beendet, wodurch auch Anwendungen aus anderen Isolationsumgebungen beeinträchtigt werden.

Status: *Schwachstelle mittels SecurityManager geschlossen*
Indem die Berechtigung RuntimePermission `exitVM` nicht an das Bundle vergeben wird, kann die Schwachstelle geschlossen werden.

Auswirkungen *Prozess*: Die Isolationsumgebung wird zerstört. Alle enthalten Anwendungen werden beendet. Das Wirtsystem oder Anwendungen aus anderen Isolationsumgebungen werden nicht beeinträchtigt.

Status: *Schwachstelle geschlossen*

ID: Mb.java.2

Quellen: [PF07]

A.1.8 Recursive Thread Creation

Ein Bundle erzeugt rekursiv Threads. Dazu wird ein Thread gestartet, der weitere drei Threads erzeugt. Diese und alle weiteren erzeugten Threads starten wieder drei weitere Threads. Jeder Thread lädt eine Datei von 225 KB Größe in den Arbeitsspeicher.

Angriffsart: Nichtverfügbarkeit

Auswirkungen *Keine Isolation*: Die OSGi-Konsole meldet eine hohe Anzahl an OutOfMemoryErrors. Das Bundle verursacht eine hohe CPU-Last und einen hohen Verbrauch an Arbeitsspeicher.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Das Bundle verursacht eine hohe CPU-Last und einen hohen Verbrauch an Arbeitsspeicher.

Status: *Schwachstelle mittels Überwachung vermeidbar*

ID: Mb.java.3

Quellen: [PF07]

A.1.9 Hanging Thread

Ein Service wird veröffentlicht, der den Thread der aufrufenden Anwendung mittels `Thread.sleep` blockiert.

Angriffsart: Nichtverfügbarkeit

Vorbedingung: Für die Untersuchung der Isolationsstufe *Keine Isolation* wird angenommen, dass sich die Bundles im Haupt-Framework befinden. Für die Stufe *Prozess* wird angenommen, dass das aufrufende Objekt zu einem Bundle gehört, das sich in einer anderen Isolationsumgebung befindet als das Bundle, das den schädlichen Service zur Verfügung stellt. Letzteres befindet sich in einer Umgebung der Stufe *Prozess*.

Der Service ist ein Kern-Service.

Auswirkungen *Keine Isolation*: Der Thread der Anwendung, die den Service aufruft, ist blockiert und kann die Ausführung nicht fortsetzen.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Der Thread der aufrufenden Anwendung wird blockiert. Wenn der Aufruf nach einer gewissen Zeit nicht beendet ist, tritt ein Timeout ein und auf Seiten der aufrufenden Anwendung wird eine Exception geworfen.

Status: *Schwachstelle geschlossen*

Implementierte Gegenmaßnahme: Für die Kommunikation über Remote Services wird vom evaluierten System R-OSGi eingesetzt. Ist ein Aufruf einer entfernten Methode nach zwei Minuten nicht beendet, wirft R-OSGi eine Exception auf Seiten der aufrufenden Anwendung.

ID: Mb.java.4

Quellen: [PF07] Das Bundle der Quelle führt die Blockierung in der Start-Methode des Activators aus. Für die Evaluierung wurde ein Bundle implementiert, das die Blockierung in einer Service-Methode durchführt.

A.1.10 Sleeping Bundle

Ein Service wird veröffentlicht, der den Thread des Aufrufenden mittels `Thread.sleep` für eine bestimmte Zeit blockiert.

Angriffsart: Leistungseinbruch

Vorbedingung: Für die Untersuchung der Isolationsstufe *Keine Isolation* wird angenommen, dass sich die Bundles im Haupt-Framework befinden. Für die Stufe *Prozess* wird angenommen, dass das aufrufende Objekt zu einem Bundle gehört, das sich in einer anderen Isolationsumgebung befindet als das Bundle, das den schädlichen Service zur Verfügung stellt. Letzteres befindet sich in einer Umgebung der Stufe *Prozess*.

Der Service ist ein Kern-Service.

Auswirkungen *Keine Isolation*: Der Thread des Aufrufenden ist für 50 Sekunden blockiert.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Der Thread des Aufrufenden ist für 50 Sekunden blockiert.

Status: *Schwachstelle offen*

Für die Kommunikation zwischen Isolationsumgebungen werden Proxy-Objekte eingesetzt. Die Proxies könnten modifiziert werden, so dass ein Aufruf nach einer bestimmten Zeit abgebrochen wird, falls er nicht beendet ist. Eine angemessene Zeitspanne zu finden, nach der jeder Aufruf beendet sein sollte, gestaltet sich jedoch als schwierig.

Die evaluierte Implementierung setzt R-OSGi für die Kommunikation ein. Bei R-OSGi tritt ein Timeout nach zwei Minuten ein, wenn ein Aufruf nicht beendet ist.

ID: Mb.java.5

Quellen: [PF07] Das Bundle der Quelle führt die Blockierung in der Start-Methode des Activators aus. Für die Evaluierung wurde ein Bundle implementiert, das die Blockierung in einer Service-Methode durchführt.

A.1.11 Big File Creator

Ein Bundle erzeugt eine große Datei, so dass weniger Massenspeicher zur Verfügung steht.

Angriffsart: Leistungseinbruch

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt Schreibrechte (FilePermission `write`). Schreibrechte für die Persistent Storage Area werden einem Bundle immer eingeräumt, da sie Teil der Implied Permissions sind [OSG09a].

Auswirkungen *Keine Isolation*: Die Datei wird in der Persistent Storage Area des Bundles erzeugt.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Die Datei wird in der Persistent Storage Area des Bundles erzeugt.

Status: *Schwachstelle mittels Überwachung vermeidbar*

Es wird ein Security Manager eingesetzt. Zusätzlich zur Persistent Storage Area werden keine Schreibrechte an das Bundle vergeben. Die Persistent Storage Area wird überwacht. Wird ein bestimmter Verbrauch an Massenspeicher überschritten, wird die Isolationsumgebung zerstört und die Persistent Storage Area gelöscht.

ID: Mb.java.6

Quellen: [PF07]

A.1.12 Code Observer

Ein Bundle untersucht Klassen eines anderen Bundles mittels Reflection. Als Startpunkt dient ein Service-Objekt des angegriffenen Bundles, dessen Class Loader verwendet wird, um weitere Klassen zu untersuchen.

Angriffsart: Unberechtigter Zugriff

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt die Berechtigungen RuntimePermission `accessDeclaredMembers` und RuntimePermission `getClassLoader`[1]. Das angegriffene Bundle registriert einen Kern-Service. Die Namen der untersuchten Klassen, außer der Klasse des Service-Objekts, sind dem Angreifer bereits vor dem Angriff bekannt.

Für die Untersuchung der Isolationsstufe *Keine Isolation* wird angenommen, dass sich die Bundles im Haupt-Framework befinden. Für die Stufe *Prozess* wird angenommen, dass sich das anzugreifende Bundle in einer anderen Umgebung befindet als das Bundle, das den Angriff mittels Reflection durchführt. Letzteres befindet sich in einer Umgebung der Stufe *Prozess*.

Auswirkungen *Keine Isolation*: Der Angreifer gibt folgende Informationen über die untersuchten Klassen aus:

- Namen der Attribute sowie deren Datentypen und Sichtbarkeitsmodifizierer
- Namen der Methoden sowie deren deren Rückgabe- und Parametertypen und Sichtbarkeitsmodifizierer
- Alle Konstruktoren sowie deren Parametertypen
- Namen der implementierten Schnittstellen
- Name des zugehörigen Pakets
- Namen der enthaltenen Klassen (inneren Klassen)
- Namen der übergeordneten Klasse in der die untersuchte Klasse enthalten ist

Die Informationen werden auch für Attribute, Methoden und Konstruktoren ausgegeben, deren Sichtbarkeit nicht als öffentlich (`public`) deklariert ist.

Die letzten beiden Punkte sind zwar implementiert, jedoch sind die untersuchten Klassen nicht verschachtelt, weshalb dieser Teil des Angriffs nicht zum Einsatz kommt.

Zusätzlich zu den Informationen über die Klassen werden auch die Werte von öffentlichen statischen (`public static`) Attributen der Klasse des Service-Objekts ausgegeben.

Status: *Schwachstelle mittels SecurityManager geschlossen*
Indem die, in den Vorbedingungen genannten, Berechtigungen nicht an das Bundle vergeben werden, kann die Schwachstelle geschlossen werden. Jedoch könnten trotzdem Informationen über öffentliche (`public`) Attribute, Methoden und Konstruktoren gewonnen werden.

Auswirkungen *Prozess*: Es werden lediglich Informationen über das Proxy-Objekt des Services ausgegeben, die keinen Rückschluss auf vertrauliche Informationen des Services zulassen.

[1] Die Berechtigung ReflectPermission wurde bei den durchgeführten Tests, entgegen der Angaben in [PF07], nicht benötigt.

Status: *Schwachstelle geschlossen*
Reflection funktioniert nicht über Prozessgrenzen hinweg.

ID: Mb.java.7

Quellen: [PF07]

A.1.13 Component Data Modifier

Ein Bundle verändert die Werte der Attribute eines Service-Objekts, das von einem anderen Bundle registriert wurde. Zudem wird ein Objekt einer Klasse erzeugt, die nicht exportiert wurde und ein Attributwert davon verändert.

Angriffsart: Unberechtigter Zugriff

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt die Berechtigungen RuntimePermission `accessDeclaredMembers` und RuntimePermission `getClassLoader`[2]. Das angegriffene Bundle registriert einen Kern-Service. Der Name der erzeugten Klasse ist dem Angreifer bereits vor dem Angriff bekannt.

Für die Untersuchung der Isolationsstufe *Keine Isolation* wird angenommen, dass sich die Bundles im Haupt-Framework befinden. Für die Stufe *Prozess* wird angenommen, dass sich das anzugreifende Bundle in einer anderen Umgebung befindet als das Bundle, das den Angriff durchführt. Letzteres befindet sich in einer Umgebung der Stufe *Prozess*.

Auswirkungen *Keine Isolation*: Der Angreifer verändert die Werte von privaten Attributen des Service-Objekts. Zudem wird ein Objekt einer nicht-exportierten Klasse erzeugt und dessen Attributwerte verändert. Der Name der Klasse ist dem Angreifer bereits im voraus bekannt.

Status: *Schwachstelle mittels SecurityManager geschlossen*
Indem die, in den Vorbedingungen genannten, Berechtigungen nicht an das Bundle vergeben werden, kann die Schwachstelle geschlossen werden.

Auswirkungen *Prozess*: Das Service-Objekt ist dem Angreifer nicht zugänglich, da Java-Reflection nicht über Prozessgrenzen hinweg funktioniert. Das angreifende Bundle wirft eine Exception (ClassNotFoundException).

Status: *Schwachstelle geschlossen*
Reflection funktioniert nicht über Prozess-Grenzen hinweg.

ID: Mb.java.8

Quellen: [PF07]

[2]Die Berechtigung ReflectPermission wurde bei den durchgeführten Tests, entgegen der Angaben in [PF07], nicht benötigt.

A.1.14 Hidden Method Launcher

Ein Bundle führt Methoden eines Objekts aus, dessen Klasse nicht exportiert wurde und das nicht als Services registriert wurde. Als Startpunkt dient ein Service-Objekt des angegriffenen Bundles, das ein Attribut mit einer Referenz auf ein Objekt besitzt, das wiederum das Objekt referenziert, dessen Methoden ausgeführt werden sollen.

Angriffsart: Unberechtigter Zugriff

Vorbedingung: Das angegriffene Bundle registriert einen Kern-Service. Die Namen der Methoden sind dem Angreifer bereits vor dem Angriff bekannt.

Für die Untersuchung der Isolationsstufe *Keine Isolation* wird angenommen, dass sich die Bundles im Haupt-Framework befinden. Für die Stufe *Prozess* wird angenommen, dass sich das anzugreifende Bundle in einer anderen Umgebung befindet als das Bundle, das den Angriff durchführt. Letzteres befindet sich in einer Umgebung der Stufe *Prozess*.

Auswirkungen *Keine Isolation*: Der Angreifer führt Methoden eines Objekts aus, dessen Klasse nicht exportiert wurde und das nicht als Services registriert wurde, und gibt deren Rückgabewerte aus. Die Klassen der Objekte wurden zwar nicht exportiert, jedoch sind die benutzten Attribute und Methoden öffentlich (`public`), deshalb kann der Angriff nicht durch den Entzug einer Berechtigung verhindert werden[3]. Dazu müsste ein spezieller Security Manager implementiert werden, der solche Angriffe erkennt.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Das Service-Objekt ist dem Angreifer nicht zugänglich, da Java-Reflection nicht über Prozessgrenzen hinweg funktioniert. Das angreifende Bundle wirft eine Exception (NoSuchFieldException).

Status: *Schwachstelle geschlossen*
Reflection funktioniert nicht über Prozess-Grenzen hinweg.

ID: Mb.java.9

Quellen: [PF07]

A.1.15 Memory Load Injection

Ein Bundle verbraucht eine große Menge an Arbeitsspeicher, indem beim Start des Bundles ein großes Array angelegt wird.

[3]Die Berechtigung ReflectPermission wurde bei den durchgeführten Tests, entgegen der Angaben in [PF07], nicht benötigt.

Angriffsart: Leistungseinbruch

Auswirkungen *Keine Isolation*: Das Bundle verbraucht zusätzlich zu dem Overhead der Laufzeitumgebung ca. 244 MB an Arbeitsspeicher.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: siehe Auswirkungen für Stufe *Keine Isolation*

Status: *Schwachstelle mittels Überwachung vermeidbar*
Durch die Beschränkung der Größe des Heaps kann die Schwachstelle geschlossen werden (JRE-Kommandozeilenparameter `-Xmx`*n*). Beschränkt man die Heap-Größe z. B. auf 128MB (`-Xmx128m`) bricht das Kind-Framework den Start des Bundles ab.

ID: Mb.java.10

Quellen: [PF07]

A.1.16 Stand Alone Infinite Loop

Ein Bundle startet einen Thread, der eine Endlosschleife ausführt.

Angriffsart: Leistungseinbruch

Auswirkungen *Keine Isolation*: Das Bundle verursacht eine hohe CPU-Last.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: siehe Auswirkungen für Stufe *Keine Isolation*

Status: *Schwachstelle mittels Überwachung vermeidbar*

ID: Mb.java.11

Quellen: [PF07]

A.1.17 Infinite Loop in Method Call

Ein Bundle stellt einen Service bereit, der eine Methode enthält, die eine Endlosschleife ausführt.

Angriffsart: Nichtverfügbarkeit, Leistungseinbruch

Vorbedingung: Für die Untersuchung der Isolationsstufe *Keine Isolation* wird angenommen, dass sich die Bundles im Haupt-Framework befinden. Für die Stufe *Prozess* wird angenommen, dass das aufrufende Objekt zu einem Bundle gehört, das sich in einer anderen Isolationsumgebung befindet als das Bundle, das den schädlichen Service zur Verfügung stellt. Letzteres befindet sich in einer Umgebung der Stufe *Prozess*. Der Service ist ein Kern-Service.

Auswirkungen *Keine Isolation*: Der Thread der Anwendungen, die den Service aufruft, wird blockiert. Durch die Endlosschleife wird eine hohe CPU-Last verursacht.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Der Thread der aufrufenden Anwendung wird blockiert. Durch die Endlosschleife wird eine hohe CPU-Last verursacht. Wenn der Aufruf nach einer gewissen Zeit nicht beendet ist, tritt ein Timeout ein und auf Seiten der aufrufenden Anwendung wird eine Exception geworfen. Die hohe CPU-Last wird weiterhin verursacht.

Status: *Schwachstelle mittels Überwachung vermeidbar*
Eine endlose Blockierung der aufrufenden Anwendung kann nicht mehr auftreten. Die hohe CPU-Last kann durch das Zerstören der Isolationsumgebung vermieden werden.

Implementierte Gegenmaßnahme: Für die Kommunikation über Remote Services wird von der evaluierten Implementierung R-OSGi eingesetzt. Ist ein Aufruf einer entfernten Methode nach zwei Minuten nicht beendet, wirft R-OSGi eine Exception auf Seiten der aufrufenden Anwendung.

ID: Mb.java.12

Quellen: [PF07] Das Bundle der Quelle führt die Blockierung in der Start-Methode des Activators aus. Für die Evaluierung wurde ein Bundle implementiert, das die Blockierung in einer Service-Methode durchführt.

A.1.18 Exponential Object Creation

Ein Bundle erzeugt rekursiv Objekte. Dazu wird ein Objekt erzeugt, das im Konstruktor drei weitere Objekte erzeugt. Diese und alle weiteren erzeugten Objekte erzeugen wieder drei weitere Objekte.

Angriffsart: Nichtverfügbarkeit

Auswirkungen *Keine Isolation*: Auf der OSGi-Konsole wird wenige Sekunden nach dem Start des Bundles eine Fehlermeldung ausgegeben, dass das Bundle aufgrund eines StackOverflowErrors nicht gestartet werden kann. Das Framework läuft ohne Beeinträchtigung weiter.

Status: *Schwachstelle geschlossen*

Auswirkungen *Prozess*: Bei der Installation des Bundle wird eine BundleException geworfen, die als Ursache einen StackOverflowError angibt. Das Kind-Framework wird nicht beeinträchtigt.

Status: siehe Status von Isolationsstufe *Keine Isolation*

ID: Mb.java.13

Quellen: [PF07]

A.1.19 Launch a hidden Bundle

Die JAR-Datei eines Bundles enthält ein weiteres verstecktes Bundle, das installiert und gestartet wird.

Angriffsart: Unberechtigter Zugriff

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt die Berechtigung AdminPermission `LIFECYCLE`.

Auswirkungen *Keine Isolation*: Das versteckte Bundle wird installiert und gestartet.

Status: *Schwachstelle mittels SecurityManager geschlossen*

Auswirkungen *Prozess*: Das versteckte Bundle wird installiert und gestartet.

Status: *Schwachstelle mittels SecurityManager geschlossen*

ID: Mb.osgi.6

Quellen: [PF07]

A.1.20 Pirat Bundle Manager

Ein Bundle nutzt die Verwaltungsdienste des OSGi-Frameworks, um ein ein anderes Bundle zu starten oder zu stoppen.

Angriffsart: Unberechtigter Zugriff

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt die Berechtigung AdminPermission `LIFECYCLE`.

Für die Untersuchung der Isolationsstufe *Keine Isolation* wird angenommen, dass sich die Bundles im Haupt-Framework befinden. Für die Stufe *Prozess* wird angenommen, dass sich das anzugreifende Bundle in einer anderen Umgebung befindet als das Bundle, das den Angriff durchführt. Letzteres befindet sich in einer Umgebung der Stufe *Prozess*.

Auswirkungen *Keine Isolation*: Das angegriffene Bundle wird gestoppt und anschließend wieder gestartet.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Der Angreifer kann auf das Bundle nicht zugreifen.

Status: *Schwachstelle geschlossen*

Der Angreifer kann auf das Bundle nicht zugreifen, da sich das angegriffene Bundle in einer anderen Isolationsumgebung befindet und die Verwaltungsdienste von OSGi nur lokal funktionieren.

ID: Mb.osgi.7

Quellen: [PF07]

A.1.21 Cycle Between Services

Ein Bundle stellt einen Service *a* bereit, der von einem anzugreifenden Service *b* eines anderen Bundles aufgerufen wird (z. B. im Sinne des Whiteboard-Pattern). Statt einen Rückgabewert zu liefern, ruft *a* eine Methode von *b* auf, die wiederum zum einem Aufruf des angreifenden Service *a* führt. Dadurch entsteht eine endloser Kreislauf an Aufrufen. Ausgelöst wird der Kreislauf durch den Aufruf des Services *b* durch ein beliebiges anderes Bundle.

Angriffsart: Nichtverfügbarkeit

Vorbedingung: Für die Untersuchung der Isolationsstufe *Keine Isolation* wird angenommen, dass sich die Bundles im Haupt-Framework befinden. Im Fall der Isolationsstufe *Prozess* befindet sich das angreifende Bundle in einer Umgebung dieser Stufe. Das angegriffene Bundle befindet sich in einer anderen Umgebung, in der sich auch das Bundle befindet, das den Kreislauf durch den Aufruf des Service *b* auslöst.

Die beteiligten Services sind Kern-Services.

Auswirkungen *Keine Isolation*: Wenige Sekunden nach dem Aufruf von Service *b* wird ein StackOverflowError geworfen, wodurch der Aufruf des Service *b* nicht vollständig durchgeführt werden kann.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Zwei Minuten nach dem Aufruf von Service *b* wird eine RemoteOSGiException geworfen, wodurch der Aufruf des Service *b* nicht vollständig durchgeführt werden kann.

Status: *Schwachstelle offen*

ID: Mb.osgi.9

Quellen: [PF07]

A.1.22 Numerous Service Registration

Ein Bundle registriert eine hohe Anzahl an Services (1.000.000).

Angriffsart: Leistungseinbruch

Auswirkungen *Keine Isolation*: Beim Stoppen des Bundles wird die OSGi-Konsole für mehrere Minuten blockiert ($>$ 20 min.). Während dieser Zeit tritt eine hohe CPU-Last auf.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Wenn die Deaktivierung des Bundles nach einer gewissen Zeit nicht beendet ist, tritt ein Timeout ein und das System meldet einen Fehler beim Stoppen des jeweiligen Bundles. Es wird weiterhin eine hohe CPU-Last verursacht.

Status: *Schwachstelle mittels Überwachung vermeidbar*
Eine Blockierung des Systems kann nicht mehr auftreten. Die hohe CPU-Last kann durch das Zerstören der Isolationsumgebung vermieden werden.

Implementierte Gegenmaßnahme: Die Deaktivierung des Bundles wird in einem eigenen Thread angestoßen. Falls sie nach einer bestimmten Zeit nicht beendet ist, wird eine Fehlermeldung ausgegeben.

ID: Mb.osgi.10

Quellen: [PF07] Die Implementierung der Quelle registriert nur 100.000 Services.

A.1.23 Ausführung von schädlichem nativem Code

Ein Bundle enthält eine native Programmbibliothek, die genutzt wird um eine Datei zu löschen.

Angriffsart: Unberechtigter Zugriff

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt die Berechtigung Programmbibliotheken zu laden (RuntimePermission `loadLibrary`).

Auswirkungen *Keine Isolation*: Die Datei wird durch das Bundle gelöscht.

Status: *Schwachstelle mittels SecurityManager geschlossen*

Auswirkungen *Prozess*: Die Datei wird durch das Bundle gelöscht.

Status: *Schwachstelle mittels SecurityManager geschlossen*

ID: da.nativecode

Quellen: [PF07] Die Implementierung dieser Schwachstelle basiert auf der Implementierung der Schwachstelle *CPU Load Injection* (Mb.native.2) der Quelle

A.1.24 Blockieren der graphischen Benutzungsoberfläche

Ein Bundle blockiert die graphische Benutzungsoberfläche eines anderen Bundles durch wiederholte Aufrufe von `Thread.sleep` im Thread von AWT.

Angriffsart: Nichtverfügbarkeit

Vorbedingung: Die zu blockierenden Bundles setzen AWT zur Darstellung der graphischen Oberfläche ein.

Für die Untersuchung der Isolationsstufe *Keine Isolation* wird angenommen, dass sich die Bundles im Haupt-Framework befinden. Für die Stufe *Prozess* wird angenommen, dass sich das anzugreifende Bundle in einer anderen Umgebung befindet als das Bundle, das den Angriff durchführt. Letzteres befindet sich in einer Umgebung der Stufe *Prozess*.

Auswirkungen *Keine Isolation*: Die Benutzungsoberfläche des Bundles reagiert nicht mehr auf Eingaben.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Die Benutzungsoberfläche des angegriffenen Bundles wird nicht beeinträchtigt.

Status: *Schwachstelle geschlossen*
Da sich der Angreifer in einem eigenen Prozess befindet, kann er nur den AWT-Thread der eigenen Isolationsumgebung beeinflußen.

ID: da.blockgui

A.1.25 Unerlaubte Netzwerkzugriffe

Ein Bundle öffnet eine HTTP-Verbindung zu einem entfernten Rechner und lädt eine Datei herunter.

Angriffsart: Unberechtigter Zugriff

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt die Berechtigung zum Aufbau von Netzwerkverbindungen (SocketPermission `connect,resolve`).

Auswirkungen *Keine Isolation*: Das Bundle lädt die Datei herunter und zeigt ihren Inhalt auf der Standardausgabe an.

Status: *Schwachstelle offen*

Auswirkungen *Prozess*: Das Bundle lädt die Datei herunter und zeigt ihren Inhalt auf der Standardausgabe an.

Status: *Schwachstelle mittels SecurityManager geschlossen*

ID: da.netio

A.1.26 Unerlaubte Zugriffe auf das Dateisystem

Ein Bundle liest den Inhalt einer Datei des Dateisystems und überschreibt anschließend deren Inhalt.

Angriffsart: Unberechtigter Zugriff

Vorbedingung: Es wird kein SecurityManager verwendet oder das Bundle besitzt Schreib- und Leserechte für die Datei (FilePermission `read, write`)

Auswirkungen *Keine Isolation*: Der Inhalt der Datei wird auf der Standardausgabe ausgegeben und dann überschrieben.

Status: *Schwachstelle mittels SecurityManager geschlossen*

Auswirkungen *Prozess*: Der Inhalt der Datei wird auf der Standardausgabe ausgegeben und dann überschrieben.

Status: *Schwachstelle mittels SecurityManager geschlossen*

ID: da.fileio

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ALRL04] A. Avižienis, J.-C. Laprie, B. Randell, C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[And04] D. P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pp. 4–10. IEEE Computer Society, Washington, 2004.

[AOS06] H. Ahn, H. Oh, C.-O. Sung. Towards reliable OSGi framework and applications. In *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 1456–1461. ACM, New York, 2006.

[Apaa] Apache Software Foundation. Apache CXF Distributed OSGi. URL http://cxf.apache.org/dosgi-discovery.html.

[Apab] Apache Software Foundation. Apache MINA. URL http://mina.apache.org/.

[BCR05] N. Beldiceanu, M. Carlsson, J.-X. Rampon. Global Constraint Catalog. SICS Technical Report T2005:08, Swedish Institute of Computer Science, 2005.

[BDM09] G. Blake, R. Dreslinski, T. Mudge. A survey of multicore processors. *IEEE Signal Processing Magazine*, 26(6):26–37, 2009.

[Bel01] N. Beldiceanu. Pruning for the Minimum Constraint Family and for the Number of Distinct Values Constraint Family. In *CP '01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pp. 211–224. Springer, Berlin/Heidelberg, 2001.

[BLMM94] T. Berners-Lee, L. Masinter, M. McCahill. Uniform Resource Locators (URL). RFC 1738 (Proposed Standard), 1994. URL http://www.ietf.org/rfc/rfc1738.txt. Obsoleted by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986.

[Bor07] S. Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, pp. 746–749. ACM, New York, 2007.

[CCEB03] A. Chien, B. Calder, S. Elbert, K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5):597–610, 2003.

[CCWY05] B. Calder, A.-A. Chien, J. Wang, D. Yang. The entropia virtual machine for desktop grids. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pp. 186–196. ACM, New York, 2005.

[CD01] G. Czajkowski, L. Daynés. Multitasking without comprimise: a virtual machine evolution. In *Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 125–138. ACM, New York, 2001. doi:http://doi.acm.org/10.1145/504282.504292.

[CDF+05] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Néri, O. Lodygensky. Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems*, 21(3):417 – 437, 2005. P2P computing and interaction with grids.

[cho] Choco Constraint Solver. URL http://choco.emn.fr/.

[Choo8] Choco Team. choco: an Open Source Java Constraint Programming Library. In *Proceedings of the Third International CSP Solver Competition*. 2008.

[CKB+07] S.-J. Choi, H.-S. Kim, E.-J. Byun, M.-S. Baik, S.-S. Kim, C.-Y. Park, C.-S. Hwang. Characterizing and Classifying Desktop Grid. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pp. 743–748. IEEE Computer Society, Washington, 2007.

[CLW96] B. M. W. Cheng, J. H. M. Lee, J. Wu. Speeding Up Constraint Propagation By Redundant Modeling. In *2nd International Conference on Principles and Practice of Constraint Programming*, pp. 91–103. Springer, Berlin/Heidelberg, 1996.

[CSP08] CSP Solver Competition. Ergebnisse der Third International CSP Solver Competition. 2008. URL http://www.cril.univ-artois.fr/CPAI08/results/results.php?idev=15.

[Die06] R. Diestel. Graphentheorie. 2006.

[Ecla] Eclipse Foundation. Eclipse Communication Framework. URL http://www.eclipse.org/ecf/.

[Eclb] Eclipse Foundation. Eclipse Rich Client Platform. URL http://wiki.eclipse.org/index.php/Rich_Client_Platform.

[Eclc] Eclipse Foundation. Equinox OSGi Framework. URL http://www.eclipse.org/equinox/.

[GE03] L. Gong, G. Ellison. *Inside Java 2 Platform Security: Architecture, API Design, and Implementation*. Pearson Education, second edition, 2003.

[GRD08] K. Gama, W. Rudametkin, D. Donsez. Using fail-stop proxies for enhancing services isolation in the OSGi service platform. In *Proceedings of the 3rd workshop on Middleware for service oriented computing*, pp. 7–12. ACM, New York, 2008.

[GTFC08] N. Geoffray, G. Thomas, B. Folliot, C. Clément. Towards a new isolation abstraction for OSGi. In *Proceedings of the 1st workshop on Isolation and integration in embedded systems*, pp. 41–45. ACM, New York, 2008.

[GTM+09] N. Geoffray, G. Thomas, G. Muller, P. Parrend, S. Frénot, B. Folliot. I-JVM: a Java Virtual Machine for Component Isolation in OSGi. Research Report RR-6801, INRIA, 2009.

[Hebo8] E. Hebrard. Mistral, a constraint satisfaction library. In *Proceedings of the Third International CSP Solver Competition*. 2008.

[KWBT] S. Krause, C. Whipkey, C. D. Bennett, The Anh Tran. Algorithmus zur Darstellung der Mandelbrot-Menge. URL http://shootout.alioth.debian.org/u64q/program.php?test=mandelbrot&lang=java&id=3.

[LB98] S. Liang, G. Bracha. Dynamic Class Loading in the Java Virtual Machine. In *OOPSLA '98: Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 36–44. ACM, New York, 1998.

[Lina] Linux Man Page. *ps*. URL http://linux.die.net/man/1/ps.

[Linb] Linux Man Page. *top*. URL http://linux.die.net/man/1/top.

[Linc] Linux Man Page. *ulimit*. URL http://linux.die.net/man/1/ulimit.

[LL06] Y. C. Law, J. H. Lee. Symmetry Breaking Constraints for Value Symmetries in Constraint Satisfaction. *Constraints*, 11(2-3):221–267, 2006.

[MKFL08] A.-C. Marosi, P. Kacsuk, G. Fedak, O. Lodygensky. Using Virtual Machines in Desktop Grid Clients for Application Sandboxing. Technical Report TR-140, CoreGRID, 2008.

[MMJZ09] W. Mao, A. Martin, H. Jin, H. Zhang. Innovations for Grid Security from Trusted Computing. In *Security Protocols: 14th International Workshop*, volume 5087 of *Lecture Notes in Computer Science*, pp. 132–149. Springer, Berlin/Heidelberg, 2009.

[NSBK03] V.-K. Naik, S. Sivasubramanian, D. Bantz, S. Krishnan. Harmony: A Desktop Grid for Delivering Enterprise Computations. In *Proceedings of the 4th International Workshop on Grid Computing*, p. 25. IEEE Computer Society, Washington, 2003.

[Opeoo] Open Services Gateway Initiative. OSGi Service Platform - Core Specification. Release 1.0. 2000.

[OSG] OSGi Alliance. Website von OSGi. URL http://www.osgi.org/.

[OSGo9a] OSGi Alliance. OSGi Service Platform - Core Specification. Release 4, Version 4.2. 2009.

[OSGo9b] OSGi Alliance. OSGi Service Platform - Service Compendium. Release 4, Version 4.2. 2009.

[OSGo9c] OSGi Alliance. RFC 112 - Bundle Repository Description. *OSGi Service Platform Release 4 Version 4.2*, Early Draft 3, 2009.

[OSGo9d] OSGi Alliance. RFC 138 - Multiple Frameworks In One JVM. *OSGi Service Platform Release 4 Version 4.2*, Early Draft 3, 2009.

[PCW89] D. L. Parnas, P. C. Clements, D. M. Weiss. Enhancing reusability with information hiding. pp. 141–157, 1989.

[PF07] P. Parrend, S. Frénot. Java Components Vulnerabilities - An Experimental Classification Targeted at the OSGi Platform. Research Report 6231, INRIA, 2007.

[RAR07] J.-S. Rellermeyer, G. Alonso, T. Roscoe. R-OSGi: distributed applications through software modularization. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, pp. 1–20. Springer, New York, 2007.

[RBW06] F. Rossi, P. van Beek, T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, 2006.

[SBHD08] S. Schulz, W. Blochinger, M. Held, C. Dangelmayr. COHESION – A microkernel based Desktop Grid platform for irregular task-parallel applications. *Future Generation Computer Systems*, 24(5):354–370, 2008.

[Sch01] U. Schöning. *Theoretische Informatik*. Spektrum Akademischer Verlag, Heidelberg Berlin, vierte edition, 2001.

[Sun06a] Sun Microsystems. Extension Mechanism Architecture. *Java SE 6 Documentation*, 2006. URL http://java.sun.com/javase/6/docs/technotes/guides/extensions/spec.html.

[Sun06b] Sun Microsystems. Java Platform Standard Edition 6 API Specification. *Java SE 6 Documentation*, 2006. URL http://java.sun.com/javase/6/docs/api/index.html.

[Sun06c] Sun Microsystems. JSR 121: Application Isolation API Specification (Final Release). 2006. URL http://jcp.org/en/jsr/detail?id=121.

[Tam] N. Tamura. Cream: Class Library for Constraint Programming in Java. URL http://bach.istc.kobe-u.ac.jp/cream/.

[Tsa93] E. Tsang. *Foundations of Constraint Programming*. Academic Press, London, San Diego, 1993.

[TTB08] T. Tanjo, N. Tamura, M. Banbara. Sugar++: A SAT-Based MAX-CSP/COP Solver. In *Proceedings of the Third International CSP Solver Competition*. 2008.

[VH02] P. Van Hentenryck. Constraint and Integer Programming in OPL. *INFORMS Journal on Computing*, 14(4):345–372, 2002.

[VNE+08] G. Valleé, T. Naughton, C. Engelmann, H. Ong, S. L. Scott. System-Level Virtualization for High Performance Computing. In *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp. 636–643. IEEE Computer Society, Washington, 2008. doi:http://dx.doi.org/10.1109/PDP.2008.85.

[Wal07] T. Walsh. Symmetry in Constraint Optimization. In *SymCon'07 Seventh International Workshop on Symmetry in Constraint Satisfaction Problems*. Providence, 2007.

Alle URLs wurden zuletzt am 01.03.2010 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Robert Hanussek)